

```
#include <stdio.h>
#include <stdlib.h>
void main() {
    char vName[20], pTag;
    printf("Into a name:");
    scanf("%s", &vName);
    printf("What is your tag:");
    scanf("%c", &pTag);
    if (pTag == 'a') {
        printf("You go now!! \n");
    }
    else if (destino == 2) {
        if (trecho == 1)
            printf("Regiao n");
        else
            printf("Regio");
    }
    else if (destino == 1) {
        if (trecho == 1)
            printf("Regiao n");
        else
            printf("Regio");
    }
}
```

# C

## Linguagem

### *Introdução ao C em 10 aulas*

Marcelo Otone Aguiar  
Rodrigo Freitas Silva

Alegre  
Marcelo Otone Aguiar  
2016



Marcelo Otone Aguiar  
Rodrigo Freitas Silva

# INTRODUÇÃO AO C EM 10 AULAS

1º Edição

Alegre  
Marcelo Otone Aguiar  
2016



Introdução ao C em 10 aulas de Marcelo Otone Aguiar; Rodrigo Freitas Silva está licenciado com uma Licença [Creative Commons - Atribuição-NãoComercial-SemDerivações 4.0 Internacional](https://creativecommons.org/licenses/by-nc-nd/4.0/).

Dados Internacionais de Catalogação na Publicação (CIP)  
Ficha catalográfica feita pelo autor

A282i Aguiar, Marcelo O.  
Introdução ao C em 10 aulas. / Marcelo Otone Aguiar; Rodrigo Freitas Silva. - 1. ed. - Alegre: Marcelo Otone Aguiar, 2016.

294 p.; 19 x 26,5 cm.

ISBN 978-85-922790-0-4

1. Linguagem de programação para computadores. 2. Linguagem C.  
3. Sintaxe. 4. Introdução à linguagem de programação.  
1. Título

CDD 000  
CDU 004.43C

# Lista de ilustrações

Figura 1 – Etapas da construção de uma aplicação . . . . .	10
Figura 2 – Processo de compilação e linkagem . . . . .	11
Figura 3 – Saída de exemplo da função printf() . . . . .	18
Figura 4 – Saída de exemplo da função printf() ajustada . . . . .	19
Figura 5 – Saída de exemplo do printf() com 3 parâmetros . . . . .	19
Figura 6 – Saída de exemplo com dois printf() . . . . .	20
Figura 7 – Saída de exemplo com dois printf() ajustado . . . . .	20
Figura 8 – Saída de exemplo da função scanf() para texto . . . . .	23
Figura 9 – Saída de exemplo para leitura de texto composto . . . . .	23
Figura 10 – Código com indentação e sem indentação . . . . .	34
Figura 11 – Exemplo de aplicação dos operadores lógicos . . . . .	38
Figura 12 – Nº de instruções executados x Laços aninhados . . . . .	54
Figura 13 – Saída de exemplo de laço aninhado com <b>for</b> . . . . .	55
Figura 14 – Representação de um vetor de 10 posições . . . . .	79
Figura 15 – Representação de um vetor com alguns valores armazenados . . . . .	80
Figura 16 – Exemplos de matrizes . . . . .	88
Figura 17 – Exemplo 1 de tabela . . . . .	88
Figura 18 – Exemplo 2 de tabela . . . . .	88
Figura 19 – Exemplo de acesso à matriz . . . . .	90
Figura 20 – Exemplo de estrutura . . . . .	99
Figura 21 – Exemplo de vetor de estrutura . . . . .	104
Figura 22 – Função de cálculo da área do retângulo . . . . .	118
Figura 23 – Teste de mesa com a função fatorial recursiva . . . . .	130
Figura 24 – Teste de mesa com a função fatorial recursiva sem interrupção das recursões . . . . .	131
Figura 25 – Teste de mesa com a função fatorial recursiva sem interrupção das recursões . . . . .	132
Figura 26 – Representação didática de alocação dinâmica . . . . .	153
Figura 27 – Representação didática de alocação dinâmica de matriz . . . . .	160
Figura 28 – Formato de preenchimento da matriz para o exercício . . . . .	163
Figura 29 – Formato de preenchimento da matriz para o exercício . . . . .	291



## Lista de tabelas

Tabela 1 – Tipos básicos da linguagem C . . . . .	14
Tabela 2 – Situações incorretas na nomenclatura das variáveis . . . . .	15
Tabela 3 – Palavras reservadas da linguagem C . . . . .	15
Tabela 4 – Operadores aritméticos e unários . . . . .	16
Tabela 5 – Operadores de atribuição . . . . .	17
Tabela 6 – Formatadores de tipo em C . . . . .	18
Tabela 7 – Códigos de barra invertida . . . . .	21
Tabela 8 – Operadores relacionais . . . . .	31
Tabela 9 – Operadores lógicos . . . . .	36
Tabela 10 – Tamanho dos tipos básicos na linguagem C . . . . .	78



# Sumário

<b>Introdução</b>	<b>7</b>
<b>1 Introdução à Linguagem C</b>	<b>9</b>
1.1 Programação em Linguagem C	10
1.2 Meu primeiro Programa em C	12
1.3 Variáveis	13
1.4 Atribuição	15
1.5 Entrada e Saída	17
1.6 Resumo da Aula	26
1.7 Exercícios da Aula	27
<b>2 Estruturas de Decisão</b>	<b>29</b>
2.1 Estruturas de Decisão	30
2.2 Cláusula if-else	30
2.3 Indentação	33
2.4 Cláusula if-else com <b>n</b> blocos de instruções	34
2.5 Cláusula if-else com condições compostas	36
2.6 Cláusula if-else com condições aninhadas	39
2.7 Cláusula switch	41
2.8 Resumo da Aula	44
2.9 Exercícios da Aula	45
<b>3 Estruturas de Iteração</b>	<b>49</b>
3.1 Estruturas de Iteração	50
3.2 Cláusula for	50
3.3 Cláusula for com laços aninhados	54
3.4 <i>Loop</i> infinito na cláusula for	56
3.5 Cláusula while	57
3.6 Validação de dados com while	60
3.7 Cláusula while com laços aninhados	61
3.8 <i>Loop</i> infinito na cláusula while	62
3.9 Cláusula do-while	63
3.10 Exemplos adicionais	66
3.11 Resumo da Aula	69
3.12 Exercícios da Aula	70
<b>4 Vetores</b>	<b>77</b>
4.1 Vetores	78
4.2 Atribuição e obtenção de valores em vetor	79
4.3 Atribuição e acesso a valores com estrutura de iteração	81
4.4 Resumo da Aula	83
4.5 Exercícios da Aula	84
<b>5 Matrizes</b>	<b>87</b>
5.1 Matrizes	88
5.2 Dimensionando uma Matriz	89
5.3 Atribuição e obtenção de valores em matriz	90
5.4 Atribuição e acesso a valores com estrutura de iteração	91
5.5 Resumo da Aula	93
5.6 Exercícios da Aula	94
<b>6 Tipos de Dados definidos pelo Programador</b>	<b>97</b>
6.1 Tipo de dado	98

6.2	Estruturas de dados	98
6.3	Estruturas de dados em linguagem C	99
6.4	Variáveis e tipos de dados de estruturas	100
6.5	Vetores de estruturas e tipos de dados	103
6.6	Estruturas aninhadas	105
6.7	Resumo da Aula	108
6.8	Exercícios da Aula	109
<b>7</b>	<b>Funções</b>	<b>111</b>
7.1	Funções	112
7.2	Funções presentes na linguagem C	112
7.3	A forma geral de uma função	116
7.4	Protótipos de funções	118
7.5	Escopo das Variáveis	121
7.6	Resumo da Aula	123
7.7	Exercícios da Aula	124
<b>8</b>	<b>Recursividade</b>	<b>127</b>
8.1	Recursividade	128
8.2	<i>Loop</i> infinito na recursividade	130
8.3	Resumo da Aula	133
8.4	Exercícios da Aula	134
<b>9</b>	<b>Ponteiros</b>	<b>137</b>
9.1	Problema	138
9.2	Ponteiros	139
9.3	Operações com Ponteiros	141
9.4	Uso de Ponteiros com Vetores / Matrizes	143
9.5	Ponteiros Genéricos	144
9.6	Ponteiro para Ponteiro	145
9.7	Resumo da Aula	147
9.8	Exercícios da Aula	148
<b>10</b>	<b>Alocação Dinâmica de Memória</b>	<b>151</b>
10.1	Alocação Estática de Memória	152
10.2	Alocação Dinâmica de Memória	152
10.3	Implementação de Alocação Dinâmica	153
10.4	Alocação Dinâmica de Matrizes	159
10.5	Resumo da Aula	162
10.6	Exercícios da Aula	163
<b>A</b>	<b>Exercícios Resolvidos da Aula 1</b>	<b>165</b>
<b>B</b>	<b>Exercícios Resolvidos da Aula 2</b>	<b>173</b>
<b>C</b>	<b>Exercícios Resolvidos da Aula 3</b>	<b>189</b>
<b>D</b>	<b>Exercícios Resolvidos da Aula 4</b>	<b>213</b>
<b>E</b>	<b>Exercícios Resolvidos da Aula 5</b>	<b>227</b>
<b>F</b>	<b>Exercícios Resolvidos da Aula 6</b>	<b>241</b>
<b>G</b>	<b>Exercícios Resolvidos da Aula 7</b>	<b>255</b>
<b>H</b>	<b>Exercícios Resolvidos da Aula 8</b>	<b>269</b>
<b>I</b>	<b>Exercícios Resolvidos da Aula 9</b>	<b>277</b>
<b>J</b>	<b>Exercícios Resolvidos da Aula 10</b>	<b>285</b>
	<b>Referências</b>	<b>293</b>

# Introdução

**E**STE LIVRO tem por objetivo conduzir o leitor no aprendizado dos conceitos introdutórios da linguagem C ao longo de 10 aulas. Quando diz-se "conceitos introdutórios", refere-se ao mínimo necessário que se deve saber de uma linguagem de programação, antes de avançar na construção de aplicações mais complexas. Entende-se então como, "mínimo necessário", o usuário saber utilizar recursos na linguagem como: estruturas de decisão ou repetição, funções, vetores, ponteiros e alocação dinâmica.

É importante destacar que, como o objetivo deste livro é apoiar o aprendizado na linguagem C, então é uma premissa que o leitor do livro já tenha passado pelo aprendizado da lógica de programação, pois este tema não é abordado neste livro. Outro ponto importante em relação à linguagem C, é que, não é o objetivo deste livro, exaurir toda a linguagem, suas possibilidades e peculiaridades, visto que, trata-se de um livro introdutório, desta forma, serão abordados os principais conceitos e deixar o caminho traçado para que o leitor possa investigar mais a fundo os temas que lhe interessar, relacionados à linguagem.

Ao longo do livro, em cada aula, serão disponibilizados exercícios para praticar os conceitos vistos, contudo, sabe-se o quão comum é, termos dúvida em relação à resolução do exercício, desta forma, disponibilizou-se também um apêndice com a resolução dos exercícios para cada aula.

é preciso destacar dois pontos em relação ao apêndice que contém a resolução dos exercícios. Primeiro ponto, a resolução apresentada neste livro, em geral, será uma dentre várias possibilidades de resolução. Isso acontece em programação, pois em geral, um exercício, ou mesmo um caso real, tem várias possibilidades de ser resolvido, então o leitor terá que analisar a resolução apresentada aqui e fazer uma crítica em relação à forma como ele resolveu para validar se o fez corretamente, ou não.

O segundo ponto é, sugere-se que o leitor sempre tente fazer o exercício antes de olhar a resposta, por isso as respostas foram incluídas no final, pois o verdadeiro aprendizado sempre ocorrerá com o esforço e simplesmente olhar a resposta não demanda muito esforço.

Espero que a leitura seja agradável e que este livro possa contribuir com o seu conhecimento.

**Prof. Marcelo Otone Aguiar**



# Introdução à Linguagem C

## Metas da Aula

1. Discutir um pouco sobre os programas em linguagem C, codificação e compilação.
2. Entender e praticar os conceitos da sintaxe utilizada na linguagem de programação C.
3. Entender e praticar os conceitos básicos relacionados à linguagem, como: definição e uso de variáveis, operadores aritméticos e operações de entrada e saída.

## Ao término desta aula, você será capaz de:

1. Escrever programas em linguagem C que sejam capazes de resolver problemas simples que envolvam, por exemplo, um cálculo matemático ou a entrada e saída de informações.
2. Escrever programas que manipulem informações em variáveis.

## 1.1 Programação em Linguagem C

Para iniciar no mundo da Linguagem C, convido você leitor a primeiro entender um pouco de como funciona o processo de construção de um programa. Geralmente, este processo é similar para qualquer linguagem de programação. Em geral, o ciclo de construção de uma aplicação engloba quatro etapas (DAMAS, 2007), conforme apresentado na figura 1.

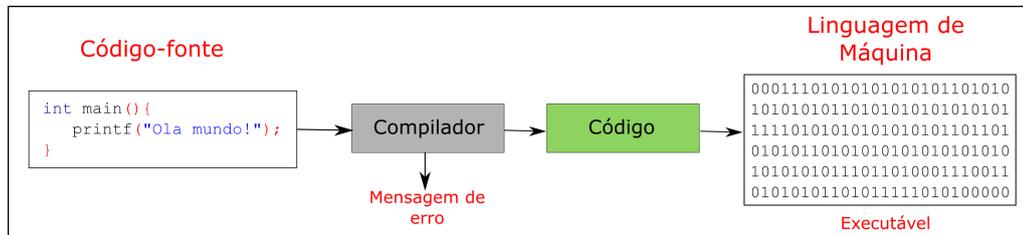


Figura 1 – Etapas da construção de uma aplicação

### 1.1.1 Etapa 1: Escrita do código-fonte

É nesta etapa que o programador realiza o esforço de escrever o código que dará origem ao programa final, conforme pode ser visto na Figura 1.1. Este código deve seguir regras rígidas para que o compilador tenha sucesso ao construir o programa. Este conjunto de regras ou formato que deve ser seguido é denominado como "**sintaxe**" da linguagem. O objetivo deste livro é orientar o leitor neste momento em que o código-fonte é escrito, ou seja, como escrever seguindo a sintaxe correta da linguagem C.

### 1.1.2 Etapa 2: Compilação do programa

Após escrever o código-fonte, o programador deve acionar o compilador para que o mesmo possa verificar a consistência do código escrito e construir o programa executável. Esta etapa é denominada "**Compilação**", e neste momento podem ocorrer duas situações: Na primeira situação, o compilador não encontra erros no código-fonte e avança para etapa seguinte. Na segunda situação o compilador encontra erros, neste caso, a operação é abortada e são exibidas mensagens com os erros que foram encontrados.

O programador deve então analisar as mensagens para corrigir os erros no código-fonte e refazer o processo de compilação. O compilador pode identificar também código-fonte que, apesar de não possuir erros na sintaxe, mas que levantam suspeita de que, ao executar o programa, o mesmo irá se comportar de forma inesperada em determinadas situações. Neste caso, o compilador avança para a próxima etapa, contudo, ele exibe avisos dos possíveis problemas, esses avisos são apresentados com o indicativo: "**Warning**".

Caso não tenha detectado erros na sintaxe (mesmo tendo emitido *Warnings*), o compilador avança para a próxima etapa e gera então, um arquivo objeto, com o nome igual ao do programa e com extensão ".o" (a extensão pode variar de um sistema operacional para outro). Para compilar um programa em linguagem C, você pode utilizar uma IDE de desenvolvimento, como: **CodeBlocks**, **DevC++**, **Netbeans**, dentre outras. Em geral, as IDE's <sup>1</sup> disponibilizam o comando para acionar o compilador de

<sup>1</sup> *Integrated Development Environment* ou Ambiente de Desenvolvimento Integrado

forma simples. Porém, o compilador pode também ser acionado por comando em modo texto. O comando para acionar o compilador irá variar de acordo com o sistema operacional e/ou compilador utilizado, conforme apresentado a seguir:

- **GCC (Windows e Linux)**  
`gcc nomePrograma.c`
- **Turbo C (Borland)**  
`tcc nomePrograma.c`
- **Borland C (Borland)**  
`bcc nomePrograma.c`
- **Microsoft C (Microsoft)**  
`cl nomePrograma.c`

### 1.1.3 Etapa 3: "Linkagem" dos objetos

A compilação faz então a verificação sintática e cria o arquivo com código-objeto. Mas o arquivo executável é criado na terceira etapa em que ocorre a "linkagem" dos objetos. O responsável por este processo é o **linker** que utiliza o arquivo objeto e as bibliotecas nativas do C, que contêm as funções já disponibilizadas pelo C, como: `printf()`, `scanf()`, `fopen()`, etc.

Basicamente o linker faz a junção dos arquivos objetos gerados e originados das bibliotecas nativas do C, em um único arquivo, o executável, conforme apresentado na figura 2.

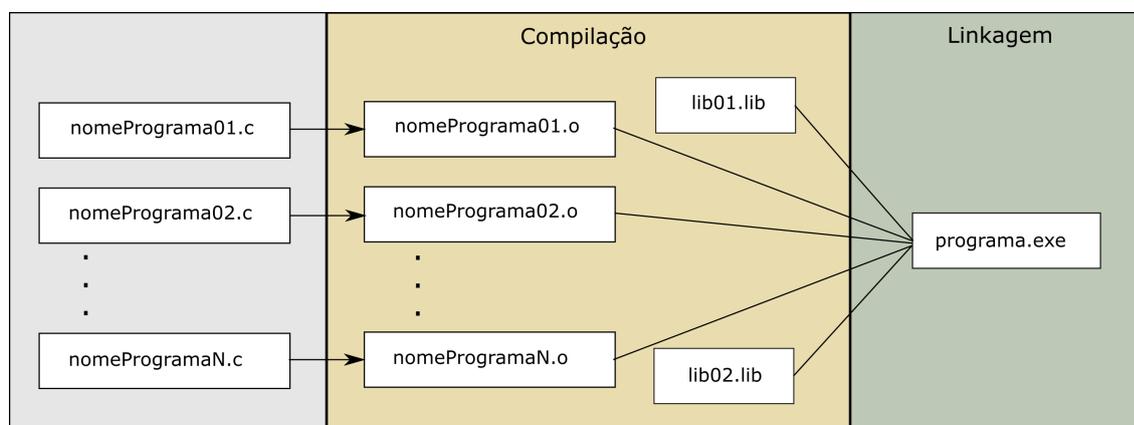


Figura 2 – Processo de compilação e linkagem

Fonte: (DAMAS, 2007, p. 5)

Normalmente, quando o compilador é acionado, este já se encarrega de acionar o processo de linkagem e não havendo erros o programa executável é construído. É comum o compilador oferecer parâmetros para serem informados, caso o programador não queira que o mesmo acione o linker automaticamente.

### 1.1.4 Etapa 4: Execução do Programa

Se as etapas de 1 a 3 ocorrerem sem erros, então ao final deste processo tem-se o arquivo executável do programa. Para testar então a aplicação criada, basta que execute

o programa da mesma maneira que executa-se outros programas no computador. No Windows, por exemplo, é só abrir a pasta em que o arquivo foi criado e executar um duplo clique no arquivo.

Essas etapas são um ciclo<sup>2</sup> que só termina quando o programa é concluído, ou seja, se ao testar o programa o comportamento não esteja dentro do esperado, basta repetir o ciclo, ajustando o código-fonte do programa e acionando o compilador.

## 1.2 Meu primeiro Programa em C

Agora que explicou-se como funciona o ciclo de construção de um aplicativo, pode-se dar os primeiros passos e construir o primeiro programa em C. Como mencionado antes, para que o compilador tenha sucesso em construir o executável, é preciso escrever uma sequência de código organizado e coeso de tal forma que seja capaz de resolver um problema logicamente. Outra questão importante que deve ser levada em consideração, é que um programa pode ser desenvolvido em módulos distintos e/ou subprogramas, assim, é necessário indicar qual o ponto de partida do programa, ou seja, em que ponto do código-fonte o programa deve iniciar suas instruções. Ao iniciar o desenvolvimento de uma aplicação em linguagem C, comumente utiliza-se o código-fonte apresentado a seguir como ponto de partida.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 void main() {
4
5 }
```

Código fonte principal em Linguagem C

Para entender o código acima, utilize a numeração que foi colocada propositalmente nas linhas do código-fonte. As linhas 1 e 2, são diretivas responsáveis pela inclusão das duas principais bibliotecas da linguagem C. Neste momento, este tema não será aprofundado, por hora, apenas entenda que essas linhas são importantes, pois são responsáveis por duas bibliotecas necessárias a um programa C, mesmo que este seja básico. A linha 3 é a definição do método **main** da aplicação, que é responsável pela definição de quais serão as primeiras instruções do programa a serem executadas, desta forma, esta é a função que se encarrega de indicar aonde o programa deve começar. Toda aplicação em linguagem C deverá possuir ao menos um método **main** para desempenhar este papel (LAUREANO, 2005). Note que, as instruções iniciais deverão ser colocadas na linha 4, ou seja, entre a linha 3 e a linha 5. Essas duas chaves "{}" encontradas nessas duas linhas são as responsáveis por indicar aonde começa e aonde termina o método **main**. Desta forma, várias instruções podem ser adicionadas no método **main** desde que estejam entre estas duas chaves.

Na linguagem C, as chaves sempre irão indicar um bloco de códigos, ou seja, sempre que desejar indicar o local de início e término de um bloco de códigos, utiliza-se chaves. Ao longo do livro várias situações serão apresentadas, em que as chaves serão utilizadas, como nas estruturas de decisão, repetição, funções, entre outras. Assim, serão várias oportunidades para entender claramente a forma de uso. Para concluir sobre o código-fonte já apresentado, tenha em mente que a partir deste ponto do livro,

<sup>2</sup> O ciclo de construção de uma aplicação apresentado aqui é aplicável a grande maioria das linguagens. No caso do C, há uma diferença em relação à compilação, pois antes mesmo de compilar é executada uma etapa em que todas as Macros são expandidas e as linhas de código são executadas para o pré-processador. Contudo, esta etapa adicional é acionada pelo próprio compilador.

este código sempre será utilizado como base para os demais exemplos e exercícios que serão realizados ao longo do aprendizado da linguagem C.

O nosso primeiro programa vai ter um objetivo muito simples, o de imprimir na tela a mensagem: "**Olá mundo!**". Para realizar essa operação precisa-se de uma função que tenha essa capacidade e a linguagem C nos disponibiliza a função **printf()**, que é, provavelmente umas das funções que mais será utilizadas ao longo do aprendizado. Para utilizar a função **printf()** corretamente, é preciso seguir a sintaxe a seguir:

```
1 //Sintaxe:
2 printf("formato", argumentos);
```

O primeiro parâmetro da função **printf()** é onde deve ser informado o texto que pretende-se apresentar na tela. Desta forma, no nosso exercício o texto "**Olá mundo!**" será informado neste parâmetro. O segundo parâmetro é opcional e, além disso, podem ser informados vários argumentos, tantos quantos forem necessários. Mas, ao informar ao menos um, o primeiro parâmetro será afetado, pois deve-se adicionar formatadores de tipo para cada argumento (LAUREANO, 2005). Por hora, este conceito não será detalhado, pois o momento mais oportuno será ao falar das variáveis.

Utilizando então a função **printf()** para fazer o nosso primeiro programa apresentar a mensagem "**Olá mundo!**", tem-se o código-fonte apresentado a seguir. Note que linha 4, em que foi adicionado a função **printf()**, não foi informado o segundo argumento da função, pois este é opcional e não é necessário quando deseja-se apenas exibir um texto fixo. Observe também que o texto está entre aspas duplas, sempre que desejar utilizar um texto em linguagem natural no código-fonte, então é necessário colocar esse texto entre aspas duplas. Por fim, observe que no final da instrução foi colocado um ponto e vírgula ";". Deve-se fazer isso sempre no final da instrução.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 void main() {
4     printf("Ola Mundo!");
5 }
```

Código-fonte do primeiro programa

## 1.3 Variáveis

Para entender claramente o papel das variáveis, basta associar aos recipientes do mundo real. Se você deseja guardar água, então provavelmente você irá utilizar uma jarra, se você deseja guardar biscoitos, então provavelmente você irá utilizar um pote hermético, se quiser guardar ovos, então irá utilizar uma caixa de ovos. Da mesma forma, utiliza-se as variáveis em um programa de computador, para guardar. A diferença é que no caso do programa, são armazenados dados, mas, assim como no mundo real, os objetos exigem recipientes diferentes e adaptados, os dados também o exigem.

Para exemplificar, se é necessário armazenar um dado que é pertencente ao conjunto dos números reais, como o valor ou o peso de um produto, então é necessário uma variável que seja compatível com este dado, ou seja, que admita valores reais, desta forma, assim como em outras linguagens, o C também requer a definição do tipo da variável. Inicialmente serão abordados apenas os tipos básicos e os mais comuns, **int**, **float**, **double** e **char**, conforme a tabela 1.

Tabela 1 – Tipos básicos da linguagem C

<i>Tipo</i>	<i>Descrição</i>
<b>int</b>	Utilizado para definir uma variável inteira que comporta valores pertencentes ao conjunto dos números inteiros.
<b>long int</b>	Utilizado para definir uma variável inteira que comporta valores maiores que o <b>int</b> .
<b>short int</b>	Utilizado para definir uma variável inteira que comporta valores menores que o <b>int</b> .
<b>float</b>	Utilizado para definir uma variável real que comporta valores pertencentes ao conjunto dos números reais.
<b>double</b>	O tipo <b>double</b> é similar ao <b>float</b> , a diferença é que o este tipo comporta valores reais com um número de dígitos maior, assim, a precisão nos cálculos com casas decimais aumenta.
<b>char</b>	Utilizado para armazenar um caractere. Comporta apenas um caractere.

Fonte: Os autores

Agora que os tipos que foram apresentados, falta entender como definir uma variável. Em linguagem C, essa tarefa é muito simples, veja a seguir a sintaxe.

```
1 //Sintaxe:
2 tipoVariavel nomeVariavel;
```

Como é apresentado na sintaxe, primeiro informa-se o tipo da variável, conforme a tabela 1, na sequência um espaço e a indicação do nome da variável. Veja a seguir alguns exemplos de definição de variáveis.

```
1 int idade;
2 float peso;
3 char genero;
4 double rendimento;
```

Na linha 1 do exemplo, foi definida uma variável, cujo nome é **idade** do tipo **int**, ou seja, é capaz de armazenar valores pertencentes ao conjunto dos números inteiros. Quando é preciso definir mais de uma variável do mesmo tipo, pode-se fazer isso em uma mesma instrução, apenas separando o nome das variáveis por vírgula, como os exemplos a seguir:

```
1 int idade, matricula;
2 float vCelcius, vKelvin;
3 char genero, sexo;
```

Assim, como em outras linguagens, o C possui várias regras para a definição de variáveis, para esclarecer o que não é permitido ao definir uma variável, veja a tabela 2, em que é apresentada na coluna da esquerda a forma não permitida em C e na coluna do meio o motivo pelo qual não é permitido e na coluna da direita uma sugestão para fazer a declaração sem erro. Além disso, a tabela 3 descreve a lista de palavras reservadas que não podem ser utilizadas na nomenclatura das variáveis.

Outro ponto que deve ser levado em consideração ao definir variáveis e que a linguagem C é "*case sensitive*", ou seja, letras maiúsculas e minúsculas correspondentes são consideradas diferentes, desta forma, ao utilizar a variável, à mesma deverá ser

Tabela 2 – Situações incorretas na nomenclatura das variáveis

<i>Forma incorreta</i>	<i>Motivo</i>	<i>Sugestão de correção</i>
<b>float</b> 4nota;	Não é permitido iniciar o nome da variável com números.	<b>float</b> nota4;
<b>char</b> float;	Não é permitido utilizar palavra reservada como nome de uma variável.	<b>char</b> vFloat;
<b>double</b> vinte%;	Não é permitido utilizar os seguintes caracteres especiais como %, @, #, \$, &, etc.	<b>double</b> vintePercent;
<b>int</b> idade pes;	Não é permitido separar os nomes compostos em variáveis.	<b>int</b> idade_pes;

Fonte: Os autores

Tabela 3 – Palavras reservadas da linguagem C

<i>Palavras Reservadas</i>					
<b>auto</b>	<b>break</b>	<b>case</b>	<b>char</b>	<b>const</b>	<b>continue</b>
<b>default</b>	<b>do</b>	<b>double</b>	<b>else</b>	<b>enum</b>	<b>extern</b>
<b>float</b>	<b>for</b>	<b>goto</b>	<b>if</b>	<b>int</b>	<b>long</b>
<b>main</b>	<b>register</b>	<b>return</b>	<b>short</b>	<b>signed</b>	<b>sizeof</b>
<b>static</b>	<b>struct</b>	<b>switch</b>	<b>typedef</b>	<b>union</b>	<b>unsigned</b>
<b>void</b>	<b>volatile</b>	<b>while</b>			

Fonte: Adaptado de Schildt (1996, p. 10)

escrita exatamente como foi escrita na definição, pois caso contrário o compilador irá entender que se trata de outra variável (PAES, 2016).

## 1.4 Atribuição

Para armazenar dados em variáveis, é preciso fazer atribuição. Para isso, deve-se utilizar o operador de atribuição, que na linguagem C é o sinal de igualdade, "=". Veja a seguir alguns exemplos do uso de atribuição.

```

1 int idade, matricula;
2 idade = 30;
3 matricula = 123659;
4 float preco = 42.9;

```

Observe que na primeira linha foram declaradas as variáveis **idade** e **matricula** do tipo **int** (inteiro). Na segunda linha o valor **30** está sendo atribuído à variável **idade**. Da mesma forma, o valor **123659** é atribuído à variável **matricula** na terceira linha. E na quarta linha, foi declarada a variável **preco** e na mesma instrução foi atribuído o valor **42.9** à variável. Todos estes exemplos são formas corretas de atribuição em C.

Além de atribuir valores simples, como exemplificado, pode-se atribuir às variáveis expressões matemáticas. Para isso, é preciso utilizar os operadores aritméticos relacionados na tabela 4. Estes operadores são nativos da linguagem C, e são a base para os cálculos mais complexos. Em geral, operadores matemáticos que não estão entre os operadores nativos da linguagem, como a raiz quadrada ou a potência, são disponibili-

zados por meio de funções programadas em C. No momento oportuno, algumas destas funções serão explicadas.

Tabela 4 – Operadores aritméticos e unários

Operador	Operação Matemática
+	Adição
-	Subtração.
*	Multiplificação
/	Divisão
%	Módulo (obtém o resto da divisão)
--	Decremento unário
++	Incremento unário

Fonte: Adaptado de [Laureano \(2005, p. 26\)](#)

Ao escrever uma expressão matemática em C é preciso considerar a ordem das operações em matemática, exemplo, na inexistência de parênteses, a operação de multiplicação e divisão será realizada antes das operações de adição e subtração. Quando houver a necessidade de alterar a precedência das operações devem-se utilizar os parênteses. Veja a seguir alguns exemplos de uso dos operadores aritméticos:

```

1 int contador = 2;
2 float valor1 = 300;
3 float valor2 = 400;
4 float totalSom = valor1 + valor2;
5 float totalMult = valor1 * valor2;
6 contador++;
7 resul = (totalMult + totalSom) * contador;

```

Nas primeiras três linhas, foi feita a declaração de variáveis e atribuição simples de valor. Na linha 4 a variável **totalSom** recebe a soma das variáveis **valor1** e **valor2**. E na linha 5 a variável **totalMult** recebe a multiplicação das variáveis **valor1** e **valor2**. Na linha 6 é utilizado o operador de incremento unário na variável **contador**, este operador incrementa o valor de uma variável a cada vez que é acionado, como a variável **contador** recebeu o valor **2** na linha 1, então, após executar a instrução na linha 6, a variável passa a ter o valor **3**. Na linha 7 a variável **resul** recebe a operação que envolve soma e multiplicação, sendo que o a operação de soma é priorizada em relação à operação de multiplicação.

É comum em programação a necessidade de acumular o valor já armazenado em uma variável ao fazer o cálculo. Neste caso, a variável recebe a própria variável acompanhada da operação de cálculo que pode ser uma soma, uma subtração, etc. Veja o exemplo de um acúmulo com soma.

```

1 float totalSom = 200;
2 float valor1 = 300;
3 float valor2 = 400;
4 totalSom = totalSom + (valor1 - valor2);

```

Note que para acumular a atribuição com o valor que já estava armazenado na variável **totalSom**, a atribuição é feita de **totalSom** adicionado da expressão matemática à **totalSom**. Assim, o valor que já estava armazenado em **totalSom** é acumulado ao invés de ser substituído. Esta operação funciona na linguagem C, contudo há uma forma mais elegante para fazê-la. Veja a seguir a mesma operação com o ajuste necessário.

```
1 float totalSom = 200;
2 float valor1 = 300;
3 float valor2 = 400;
4 totalSom += valor1 - valor2;
```

Ao analisar a linha 4 observa-se que a variável **totalSom** no lado direito da igualdade foi suprimida, pois o operador de soma antes da igualdade já indica que deseja-se que ela seja acumulada. Ao remover a variável do lado direito da igualdade, os parênteses também se tornaram desnecessários. A tabela 5 apresenta os operadores de atribuições disponíveis.

Tabela 5 – Operadores de atribuição

<i>Operador</i>	<i>Operação Matemática</i>
=	Atribuição simples
+=	Atribuição acumulando por soma.
-=	Atribuição acumulando por subtração
*=	Atribuição acumulando por multiplicação
/=	Atribuição acumulando por divisão
%=	Atribuição acumulando por módulo

Fonte: Adaptado de [Laureano \(2005, p. 27\)](#)

## 1.5 Entrada e Saída

Para um programa de computador é fundamental a interação com dispositivos de entrada e de saída, pois é por meio da entrada que os programas recebem os dados e por meio da saída que são disponibilizadas às informações aos usuários. Isso é um princípio básico da computação. Essa interação pode ocorrer com vários dispositivos, mas, neste livro, será utilizado o teclado como dispositivo de entrada e o monitor como dispositivo de saída.

### 1.5.1 Função printf()

Um pouco já foi falado dessa função no primeiro programa. Mas, agora é o momento oportuno para tratar melhor sobre ela. A função **printf()** permite realizar a impressão de textos no monitor, ou seja, é responsável pela saída de informações. Esta função possui um número variado de parâmetros, tantos quantos forem necessários. Veja a seguir a sintaxe para utilizar corretamente a função **printf()**.

```
1 //Sintaxe:
2 printf("formato", argumentos);
```

Como já mencionado antes, o primeiro argumento da função é obrigatório, ou seja, no mínimo deve-se informar um texto para ser impresso. Os próximos argumentos são opcionais, pois nem sempre é necessário apresentar uma informação em conjunto do texto. Um exemplo de como utilizar a função **printf()** apenas com o primeiro argumento, pode ser visto a seguir. O uso da função com mais de um argumento requer também o uso de formatadores de tipo, conforme a tabela 6. Veja a seguir como utilizar a função **printf()** com dois ou mais argumentos.

```

1 float total = 300 + 400;
2 printf("Total da conta: %f", total);

```

Tabela 6 – Formataadores de tipo em C

Formato	Tipo da variável	Conversão realizada
<b>%c</b>	Caracteres	char, short int, int, long int
<b>%d</b>	Inteiros	int, short int, long int
<b>%e</b>	Ponto flutuante, notação científica	float, double
<b>%f</b>	Ponto flutuante, notação decimal	float, double
<b>%lf</b>	Ponto flutuante, notação decimal	double
<b>%g</b>	O mais curto de %e ou %f	float, double
<b>%o</b>	Saída em octal	int, short int, long int, char
<b>%s</b>	String	char *, char[]
<b>%u</b>	Inteiro sem sinal	unsigned int, unsigned short int, unsigned long int
<b>%x</b>	Saída em hexadecimal (0 a f)	int, short int, long int, char
<b>%X</b>	Saída em hexadecimal (0 a F)	int, short int, long int, char
<b>%ld</b>	Saída em decimal longo	Usado quando long int e int possuem tamanhos diferentes.

Fonte: Adaptado de [Laureano \(2005, p. 21\)](#)

Observe que na segunda linha do exemplo a função **printf()** foi acionada com dois argumentos, o primeiro argumento é o texto combinado com o formatador de tipo e o segundo é a variável que foi definida na linha anterior. O objetivo é que seja impresso o texto combinado com o dado armazenado na variável. O dado armazenado na variável é o resultado da expressão **300 + 400**, portanto o dado armazenado é **700**. Assim sendo, a saída deste programa deve ser conforme apresentado na figura 3.

SAÍDA DO PROGRAMA

TOTAL DA CONTA: 700.000000

Figura 3 – Saída de exemplo da função printf()

Ou seja, o dado da variável é substituído pelo formatador de tipo **%f**. Observe que foi utilizado o formatador para ponto flutuante com notação decimal, de acordo com a tabela 6, pois a variável a ser impressa é do tipo **float** e deseja-se imprimir o dado no formato decimal. Se a variável fosse do tipo **int**, então pode-se utilizar o formatador **%d**. Nota-se também que o dado **700** foi impresso com vários zeros em sua fração, isso aconteceu porque não foi realizada uma formatação do número para indicar a quantidade de casas decimais que deseja-se.

Para formatar um **float**, basta seguir a definição:

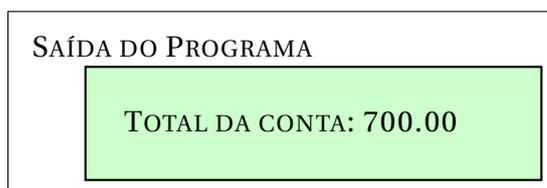
```
1 \#[tam].[casa_dec]f
```

Em que:

- **tam** – indica o tamanho mínimo que deve ser impresso na saída. Se o número possuir um tamanho superior ao informado neste parâmetro, o número não será truncado.
- **casa\_dec** – número de casas decimais que devem ser impressas. Neste caso, se o número possuir uma quantidade de casas decimais superior ao indicado, então as casas decimais serão truncadas.

Assim, se quiser imprimir a informação com apenas duas casas decimais, basta ajustar o código da seguinte forma:

```
1 float total = 300 + 400;  
2 printf("Total da conta: %3.2f", total);
```



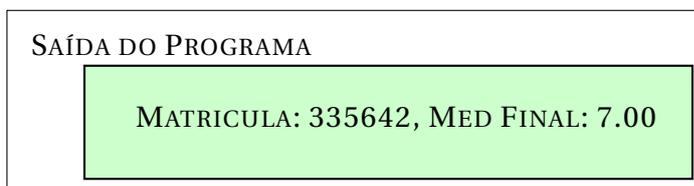
SAÍDA DO PROGRAMA

```
TOTAL DA CONTA: 700.00
```

Figura 4 – Saída de exemplo da função printf() ajustada

Veja que a saída apresentada na figura 4 obedeceu à formatação de duas casas decimais e naturalmente a informação apresentada na saída ficou mais elegante. Observe agora um exemplo de uso da função printf() com 3 parâmetros.

```
1 int mat = 335642;  
2 float medF = 7;  
3 printf("Matricula: %d, Med Final: %2.2f ", mat, medF);
```



SAÍDA DO PROGRAMA

```
MATRICULA: 335642, MED FINAL: 7.00
```

Figura 5 – Saída de exemplo do printf() com 3 parâmetros

No exemplo com três parâmetros apresentado na figura 5, observa-se que foram adicionados dois formataores no texto, o primeiro **%d**, pois o segundo parâmetro é uma variável do tipo **int** e o segundo formataor **%2.2f** visto que o terceiro parâmetro é uma variável do tipo **float**. Conclui-se então que, o texto pode ser combinado com tantas quantas forem às informações que se deseja apresentar, e que para isso, basta adicionar os formataores de tipo na mesma ordem em que os parâmetros seguintes serão informados. Veja agora outro exemplo de uso da função printf().

```
1 int mat = 335642;
2 float medF = 7;
3 printf("Matricula: %d", mat);
4 printf("Media Final: %2.2f", medF);
```

SAÍDA DO PROGRAMA

```
MATRICULA: 335642MEDIA FINAL: 7.00
```

Figura 6 – Saída de exemplo com dois printf()

Note na figura 6 como a saída do programa foi impressa. Veja que o texto impresso pela primeira função **printf()** está **concatenado** com o texto impresso pela segunda função. Isso ocorreu porque não foi informado para a primeira função a intenção de adicionar espaço, tabulação ou quebra de linha. Como é possível resolver isso então? A linguagem C tem uma forma bem elegante de resolver isso, ela trata o caractere "\n" como sendo especial para uso combinado com outros caracteres que irão permitir operações especiais no texto. Para resolver o problema do exemplo anterior, basta adicionar uma quebra de linha. Veja a seguir:

```
1 int mat =335642;
2 float medF = 7;
3 printf("Matricula: %d\n", mat);
4 printf("Media Final: %2.2f", medF);
```

SAÍDA DO PROGRAMA

```
MATRICULA: 335642
MEDIA FINAL: 7.00
```

Figura 7 – Saída de exemplo com dois printf() ajustado

Note que no exemplo foram adicionados os caracteres "\n" na primeira função **printf()**. Ao fazer isso, a função entenderá que deve "mudar de linha" após a impressão do texto anterior aos caracteres "\n", com isso, o texto da segunda função foi impresso na linha seguinte. Veja na tabela 7 outros caracteres que podem ser combinados com o caractere especial "\n" e o significado de cada um.

### 1.5.2 Função scanf()

Como visto a função **printf()** é responsável pela saída das informações do programa. Como fazer então para entrar com dados na fronteira do programa? Neste caso, deve-se utilizar a função **scanf()**. Similar à função **printf()**, a função **scanf()** também suporta uma quantidade "n" de argumentos e permite que os dados digitados pelo usuário do

Tabela 7 – Códigos de barra invertida

Caracteres	Ação
<code>\b</code>	Retrocesso ( <i>back</i> )
<code>\f</code>	Alimentação de formulário ( <i>form feed</i> )
<code>\n</code>	Nova linha ( <i>new line</i> )
<code>\t</code>	Tabulação horizontal ( <i>tab</i> )
<code>\"</code>	Aspas
<code>\'</code>	Apóstrofo
<code>\0</code>	Nulo (0 em decimal)
<code>\\</code>	Barra invertida
<code>\v</code>	Tabulação vertical
<code>\a</code>	Sinal sonoro ( <i>beep</i> )
<code>\N</code>	Constante octal (N é o valor da constante)
<code>\xN</code>	Constante hexadecimal (N é o valor da constante)

Fonte: Adaptado de Schildt (1996, p. 38)

programa sejam armazenados nas variáveis do programa. Veja a seguir a sintaxe para o uso da função `scanf()`.

```
1 //Sintaxe:
2 scanf("formato", enderecosArgumentos);
```

No caso da função `scanf()`, no mínimo devem ser informados dois argumentos, o primeiro é responsável pelo formato dos dados de entrada, e estes, seguirão o mesmo padrão apresentado na tabela 6 e o segundo argumento é o endereço da variável, para armazenar o dado digitado pelo usuário. Veja a seguir um exemplo de uso da função.

```
1 int mat;
2 scanf("%d", &mat);
```

Na linha 1 do exemplo, foi feita a declaração da variável que será utilizada para armazenar o dado. Na linha 2 foi realizado o acionamento da função `scanf()`, note que no primeiro argumento foi informado o formato entre aspas, "%d", conforme apresentado na tabela 6 para o tipo de dado inteiro. E o segundo argumento é o caractere `&` acompanhado do nome da variável. Como dito antes, a partir do segundo argumento deve-se informar o endereço de memória no qual o dado será armazenado, por isso, foi utilizado o caractere `&` acompanhado do nome da variável, pois em C, o caractere `&` é responsável por indicar que o retorno será o endereço de memória de uma determinada variável. Veja agora um exemplo do uso da função para três argumentos.

```
1 float nota1, nota2;
2 scanf("%f %f", &nota1, &nota2);
```

A primeira linha do exemplo é a declaração de duas variáveis do tipo `float` e a segunda linha é o acionamento da função `scanf()`, neste caso, para a leitura e armazenamento em duas variáveis, `nota1` e `nota2`. Note que, no formato foi escrito o "%f" duas vezes e separado por um espaço, assim, o `scanf()` saberá tratar a formatação dos dois próximos argumentos. E da mesma forma pode-se combinar o uso não só de variáveis diferentes, mas também de tipos diferentes. Agora que foi explicado em como declarar variáveis, fazer operações de atribuição e aritméticas, e utilizar funções de entrada e saída, é possível fazer o primeiro exercício.

### 1.5.2.1 Exercício de Exemplo

Faça um programa em C que receba dois números inteiros e ao final imprima a soma deles.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 void main() {
4     int soma, num1, num2;
5     printf("Informe o primeiro numero:");
6     scanf("%d", &num1);
7     printf("Informe o segundo numero:");
8     scanf("%d", &num2);
9
10    soma = num1 + num2;
11
12    printf("Resultado da soma: %d", soma);
13 }
```

As linhas 1 e 2 do programa apresentado no exercício de exemplo, são responsáveis pela inclusão das bibliotecas necessárias para o seu funcionamento. Na linha 4 foram declaradas 3 variáveis do tipo inteiro, conforme pedido no enunciado do exercício, sendo **num1** e **num2** para receber os dois valores inteiros e **soma** para armazenar o resultado da soma dos valores. Entre as linhas 5 e 8, foram utilizadas as funções **printf()** e **scanf()** com o objetivo de obter os dados do usuário, assim, o **printf()** exibe uma mensagem para orientar o usuário no preenchimento e o **scanf()** se encarregar de capturar e armazenar o dado digitado pelo usuário.

Uma vez o dado armazenado na variável, pode-se proceder com o cálculo da soma, que ocorre na linha 10 e por fim, a impressão do resultado da soma na linha 12. Desta forma, o programa contempla entrada, o processamento e saída dos dados e atende aos requisitos do enunciado do exercício de ler os dois valores inteiros e imprimir a soma deles.

### 1.5.3 Lendo texto com a função scanf()

A função **scanf()** funciona muito bem para os tipos **int**, **float**, **double**, entre outros, contudo, quando se trata do armazenamento de um texto, caso em que, deve-se utilizar o **char** com definição de tamanho, esta função apresenta um comportamento indesejável em uma situação específica. Veja a seguir um exemplo:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     char produto[30];
7     printf("Informe o nome do produto: \n");
8     scanf("%s", &produto);
9
10    printf("Produto: %s \n", produto);
11 }
```

Agora copie o código do exemplo apresentado e utilize uma IDE, como o **Code-Blocks** para compilar o código.<sup>3</sup> Após compilar, execute e para testar, informe o seguinte nome para o produto: "Arroz" e pressione **ENTER**. Se você fez exatamente como solicitado, então você obterá a saída para a execução do programa apresentada na figura 8.

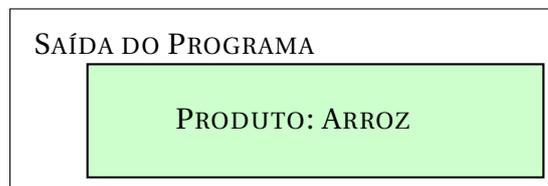
A imagem mostra uma caixa de texto com o título "SAÍDA DO PROGRAMA" no topo. Dentro da caixa, há um retângulo verde com o texto "PRODUTO: ARROZ" no centro.

Figura 8 – Saída de exemplo da função `scanf()` para texto

Agora, execute novamente o programa e digite para o nome do produto: "Arroz integral", após digitar, pressione **ENTER**. Notou que o resultado da execução é exatamente igual ao apresentado na figura 8? Isso ocorre, porque o `scanf()` não sabe lidar muito bem com nomes compostos. Para resolver isso, podemos utilizar uma formatação diferente que se encarregará de resolver este problema. Veja a seguir o código ajustado com a formatação mencionada.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     char produto[30];
7     printf("Informe o nome do produto: \n");
8     scanf("\n%[^\\n]s", &produto);
9
10    printf("Produto: %s \n", produto);
11 }
```

Veja a função `scanf()` na linha 8, notou a diferença no formato utilizado? Agora repita o teste anterior e note que o resultado será similar ao apresentado na figura 9. Contudo, esse formato não é recomendado, pois o `scanf()` poderá produzir outro efeito indesejado, no caso, tentar armazenar um texto com tamanho superior ao suportado pela variável. O método recomendado para resolver este problema é utilizar a função `fgets()`, veja na sequência, como essa função poderá ser utilizada.

A imagem mostra uma caixa de texto com o título "SAÍDA DO PROGRAMA" no topo. Dentro da caixa, há um retângulo verde com o texto "PRODUTO: ARROZ INTEGRAL" no centro.

Figura 9 – Saída de exemplo para leitura de texto composto

<sup>3</sup> Se você tiver dificuldades em executar essa operação, faça uma busca na Internet sobre o tema e veja que existem muitos sites, blogs e vídeos que ensinam um passo-a-passo.

### 1.5.4 Função `fgets()`

Outra forma de resolver o problema, apresentado pelo `scanf()` ao ler textos, é utilizar a função `fgets()`. Esta função irá armazenar corretamente o texto, mesmo quando este é composto por duas, três ou mais palavras, desde que, seja respeitado o tamanho limite determinado para o `char`. Veja a seguir o exemplo anterior, adaptado para funcionar com a função `fgets()`.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     char produto[30];
7     printf("Informe o nome do produto: \n");
8     fgets(produto, 30, stdin);
9
10    printf("Produto: %s \n", produto);
11 }
```

Veja que a única linha do código-fonte que sofreu mudança é a linha 8, na qual estava a instrução do `scanf()`, que foi substituído pelo `fgets()`. A função `fgets()` possui três argumentos, o primeiro é o nome da variável, note que, neste caso, não é necessário indicar o sinal de `&` como é necessário no `scanf()`, pois, o argumento esperado pela função `fgets()` é o nome da variável e não o endereço de memória dela. O segundo argumento é o tamanho da variável, que no exemplo, é **30**, conforme definido na linha 6, e o terceiro argumento é o **arquivo** no qual o texto deve ser armazenado, como no caso, a intenção e armazenar em uma variável na memória, então informamos a constante `stdin` para indicar que a leitura é proveniente do teclado. Se você compilar e executar agora, este trecho de código e informar o nome do produto igual a "Arroz integral", obterá a saída conforme a figura 9.

### 1.5.5 Comentários

Até este ponto do livro foi feito pouco uso dos comentários, pois a parte final da primeira aula foi reservada para falar sobre eles. Mas para que servem os comentários? Bem, até o momento, ficou claro que no contexto da programação de computadores, os compiladores são muito criteriosos e são exigentes na escrita do programa, não deixam nenhum erro sintático passar despercebido. Mas, em programas com 10 mil, 20 mil ou mais linhas de códigos, é importante, principalmente em trechos menos intuitivos, comentar (explicar) sobre o que foi escrito, mas como o compilador não aceita algo diferente da sintaxe da linguagem de programação misturada ao código, em geral, as linguagens de programação disponibilizam "**indicadores**" para de certa forma, dizer ao compilador: "**despreze esse trecho!**", assim, ao indicar qual trecho do código que o compilador deve desprezar, pode-se fazer uso deste para escrever em qualquer formato, incluindo a linguagem formal ou informal, que o compilador não irá gerar erros relacionados àquele trecho. Este recurso é chamado de "comentário".

Veja então quais são os "**indicadores**" que a linguagem C disponibiliza para fazer comentários no código. A seguir um exemplo.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 void main() {
4     int soma, num1, num2;
```

```
5 //trecho responsavel pela entrada dos dados
6 printf("Informe o primeiro numero:");
7 scanf("%d", &num1); //leitura de num1
8 printf("Informe o segundo numero:");
9 scanf("%d", &num2);
10 /* 0 trecho a seguir e responsavel pela soma dos valores de num1 e num2
    informados pelo usuario do programa */
11 soma = num1 + num2;
12
13 printf("Resultado da soma: %d", soma);
14 }
```

Há duas formas de utilizar comentários em linguagem C, a primeira forma é apresentada na linha 5 do exemplo, trata-se do uso de duas barras "//", este formato é o mais comum e permite comentar o código a partir do ponto em que as barras são incluídas e apenas na linha em que elas foram colocadas. Na linha 7 pode-se ver um exemplo de comentário que foi colocado após um trecho de código que deve ser considerado pelo compilador. Uma vantagem do uso das duas barras é que não é necessário indicar aonde termina o comentário, uma vez que, pela sua natureza apenas uma linha ou parte dela é comentada.

Contudo se for preciso escrever um comentário com mais de uma linha, então o segundo formato é o mais indicado, pois, neste caso, basta colocar o indicador de onde o comentário deve iniciar e depois colocar o indicador de onde o comentário termina, assim, é possível escrever comentários com várias linhas apenas indicando o ponto de início e término. A linha 10 apresenta o exemplo deste formato de comentário, note que neste caso, o indicador de início é "/\*" e o indicador de término é "\*/".

A partir deste ponto do livro, este recurso será frequentemente utilizado, para explicar exercícios resolvidos que forem apresentados. Pois, em alguns casos, é muito útil quando a explicação está bem ao lado do código escrito. Contudo, assim como no cotidiano de um programador, os comentários serão inseridos no código com moderação.

## 1.6 Resumo da Aula

Nesta primeira aula foram apresentados os principais conceitos introdutórios da linguagem C, que serão necessários para dar continuidade no estudo. Para desenvolver um programa básico em C, minimamente é preciso saber:

- Declarar variáveis
- Atribuir valores a variáveis
- Efetuar operações aritméticas
- Realizar entrada de dados
- Realizar saída de informações

Assim, este foi o objetivo desta aula, aprender objetivamente como realizar estas operações para que, a partir da próxima aula, seja possível aprender os conceitos mais avançados da linguagem.

No que diz respeito à declarar variáveis, é importante ficar atento ao fato de que a linguagem C é *case sensitive*, ou seja, uma variável iniciando com letra maiúscula é diferente de uma variável com mesmo nome, mas iniciando com letra minúscula, ao ficar atento a isso, em geral, você conseguirá diminuir uma boa parte dos problemas que poderá enfrentar ao se deparar com mensagens de erro apresentadas pelo compilador.

Em relação aos comandos de entrada, **scanf** e saída, **printf**, fique atento ao uso dos formatadores de tipo, pois, geralmente o seu uso incorreto é alertado pelo compilador apenas com *warnings*, mas, em geral o programa irá apresentar mal funcionamento. No caso do **scanf()**, há a necessidade também de sempre preceder o nome da variável com o caractere "&", pois, caso contrário o **scanf()** não irá funcionar corretamente.

## 1.7 Exercícios da Aula

Os exercícios desta lista foram Adaptados de [Lopes e Garcia \(2002, p. 38-52\)](#).

1. Faça um programa em C que imprima o seu nome.
2. Faça um programa em C que imprima o produto dos valores 30 e 27.
3. Faça um programa em C que imprima a média aritmética entre os números 5, 8, 12.
4. Faça um programa em C que leia e imprima um número inteiro.
5. Faça um programa em C que leia dois números reais e os imprima.
6. Faça um programa em C que leia um número inteiro e imprima o seu antecessor e o seu sucessor.
7. Faça um programa em C que leia o nome o endereço e o telefone de um cliente e ao final, imprima esses dados.
8. Faça um programa em C que leia dois números inteiros e imprima a subtração deles.
9. Faça um programa em C que leia um número real e imprima  $\frac{1}{4}$  deste número.
10. Faça um programa em C que leia três números reais e calcule a média aritmética destes números. Ao final, o programa deve imprimir o resultado do cálculo.
11. Faça um programa em C que leia dois números reais e calcule as quatro operações básicas entre estes dois números, adição, subtração, multiplicação e divisão. Ao final, o programa deve imprimir os resultados dos cálculos.
12. Faça um programa em C que leia um número real e calcule o quadrado deste número. Ao final, o programa deve imprimir o resultado do cálculo.
13. Faça um programa em C que leia o saldo de uma conta poupança e imprima o novo saldo, considerando um reajuste de 2%.
14. Faça um programa em C que leia a base e a altura de um retângulo e imprima o perímetro (base + altura) e a área (base \* altura).
15. Faça um programa em C que leia o valor de um produto, o percentual do desconto desejado e imprima o valor do desconto e o valor do produto subtraindo o desconto.
16. Faça um programa em C que calcule o reajuste do salário de um funcionário. Para isso, o programa deverá ler o salário atual do funcionário e ler o percentual de reajuste. Ao final imprimir o valor do novo salário.
17. Faça um programa em C que calcule a conversão entre graus centígrados e Fahrenheit. Para isso, leia o valor em centígrados e calcule com base na fórmula a seguir. Após calcular o programa deve imprimir o resultado da conversão.

$$F = \frac{9 \times C + 160}{5} \quad (1.1)$$

Em que:

- F = Graus em Fahrenheit
- C = Graus centígrados

18. Faça um programa em C que calcule a quantidade de litros de combustível consumidos em uma viagem, sabendo-se que o carro tem autonomia de 12 km por litro de combustível. O programa deverá ler o tempo decorrido na viagem e a velocidade média e aplicar as fórmulas:

$$D = T \times V \quad (1.2)$$

$$L = \frac{D}{12} \quad (1.3)$$

Em que:

- D = Distância percorrida em horas
- T = Tempo decorrido
- V = Velocidade média
- L = Litros de combustível consumidos

Ao final, o programa deverá imprimir a distância percorrida e a quantidade de litros consumidos na viagem.

19. Faça um programa em C que calcule o valor de uma prestação em atraso. Para isso, o programa deve ler o valor da prestação vencida, a taxa periódica de juros e o período de atraso. Ao final, o programa deve imprimir o valor da prestação atrasada, o período de atraso, os juros que serão cobrados pelo período de atraso, o valor da prestação acrescido dos juros. Considere juros simples.
20. Faça um programa em C que efetue a apresentação do valor da conversão em real (R\$) de um valor lido em dólar (US\$). Para isso, será necessário também ler o valor da cotação do dólar.

# Estruturas de Decisão

## Metas da Aula

1. Entender e praticar os conceitos da sintaxe utilizada na linguagem C para estruturas de decisão, operadores relacionais e lógicos.
2. Aplicar variadas situações relacionadas ao fluxo de decisão em programação objetivando cobrir diversificadas possibilidades vivenciadas na programação cotidiana.
3. Escrever programas que farão uso de fluxos de decisão no processamento dos dados.

## Ao término desta aula, você será capaz de:

1. Escrever programas em linguagem C que sejam capazes de resolver problemas que envolvam, por exemplo, a decisão entre um cálculo ou outro, dada uma determinada circunstância.
2. Escrever programas que manipulem as informações, considerando variados operadores relacionais e lógicos.

## 2.1 Estruturas de Decisão

Ao longo da aula 1, foram abordados os conceitos necessários para a criação de programas em C com um único fluxo ou caminho. Ou seja, programas que irão sempre executar um determinado número de instruções sempre em um mesmo formato. Contudo, em geral, uma pequena parte dos problemas computacionais se limita a problemas com tal complexidade, grande parte das vezes os problemas dão origem a programas que possuem variados fluxos ou caminhos. Para que as instruções em um programa tomem um caminho diferente, é necessário que haja uma instrução responsável pela decisão de qual caminho tomar (HASKINS, 2013; ANICHE, 2015; MANZANO, 2015).

Na linguagem C serão abordadas as instruções **if-else** e **switch**, a primeira é a mais comumente utilizada nas linguagens de programação, e a segunda, em geral, é utilizada em alguns casos específicos que serão comentados no momento oportuno. A primeira cláusula a ser abordada é o **if-else**.

## 2.2 Cláusula if-else

A cláusula **if-else** permite estabelecer um controle de fluxo no programa de forma que o mesmo, possa escolher quando executar um determinado bloco de instruções ou não, ou ainda, optar por executar um bloco de instruções em vez de outro. Veja a seguir a sintaxe:

```
1 //Sintaxe:
2 if (condicao)
3     instrucao;
4 else     //"else" e opcional
5     instrucao;
```

A sintaxe apresentada mostra a situação mais simples de uso da cláusula **if-else**, condição esta em que, apenas uma instrução será executado caso a condição seja verdadeira ou falsa. Note que o trecho da cláusula **else** é opcional, conforme o comentário apresentado. Esta cláusula só é utilizada quando é necessário pelo menos dois fluxos ou caminhos em nosso programa. Veja um exemplo em forma de exercício para entender melhor.

### 2.2.1 Exercício de Exemplo

Faça um programa em C que receba um número inteiro e verifique se este número é maior que 20, em caso afirmativo o programa deverá imprimir a mensagem: "Maior que 20".

Fonte: Adaptado de Lopes e Garcia (2002, p. 78)

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 void main() {
4     int num;
5     printf("Informe o numero:");
6     scanf("%d", &num);
7
8     if (num > 20)
9         printf("Maior que 20");
10 }
```

Para resolver este exercício foi preciso declarar uma variável para armazenar um número inteiro, realizar a leitura com a função de entrada e após isso, verificar se o número é maior que 20, para imprimir ou não a mensagem. Ou seja, a decisão no programa está relacionada a executar ou não a instrução de impressão da mensagem dada uma determinada condição que foi estabelecida como: o número é maior que 20. Até o momento em que o número é lido, o programa é similar ao que aprendido na aula 1.

Como apresentado na resposta, entre as linhas 1 e 6, o código segue a teoria e prática discutida na aula 1. Nas linhas 8 e 9, foi incluída a cláusula **if-else** em sua forma mais simples de uso, em que, a condição é simples, apenas uma instrução é executada caso essa condição seja verdadeira, e além disso, não foi necessário o uso do **else**. A linha 8 traz o início da instrução representado pela palavra reservada **if**, logo após, entre parênteses, que são obrigatórios, está a condição **num > 20**, sempre que houver uma condição, esta será acompanhada, como no exemplo, de um operador relacional, que no caso, é o sinal de maior, **>**. Este operador estabelece a relação entre os operandos. A tabela 8 apresenta os operadores relacionais disponíveis na linguagem C.

Assim, no exemplo anterior, quando o usuário do programa informar um número que seja maior que 20, a relação estabelecida na instrução da linha será verdadeira e, portanto, o comando da linha seguinte, que pertence ao bloco da cláusula **if-else**, será executado. No caso contrário, ou seja, no caso em que o usuário informa um número menor ou igual a 20, então a relação estabelecida entre os operandos será falsa e, portanto o programa irá saltar a linha 9, que pertence ao bloco da cláusula **if-else** e irá para a linha 10, não executando assim a instrução que imprime a mensagem na tela.

Tabela 8 – Operadores relacionais

<i>Operador</i>	<i>Nome</i>	<i>Exemplo</i>	<i>Significado do exemplo</i>
<code>==</code>	Igualdade	<code>a == b</code>	a é <b>igual</b> a b?
<code>&gt;</code>	Maior que	<code>a &gt; b</code>	a é <b>maior</b> que b?
<code>&gt;=</code>	Maior ou igual que	<code>a &gt;= b</code>	a é <b>maior ou igual</b> a b?
<code>&lt;</code>	Menor que	<code>a &lt; b</code>	a é <b>menor</b> que b?
<code>&lt;=</code>	Menor ou igual que	<code>a &lt;= b</code>	a é <b>menor ou igual</b> a b?
<code>!=</code>	Diferente de	<code>a != b</code>	a é <b>diferente</b> de b?

Fonte: Adaptado de (DAMAS, 2007, p. 53)

Como saber que a linha 9 pertence ao bloco de instruções da linha 8? Simples, no exemplo apresentado, não há uso de chaves após a instrução da linha 8, isso quer dizer que para este bloco de instruções do comando **if** será considerada apenas uma instrução, que é a próxima, no caso, a linha 9 do código. Mas, e se for necessário que o bloco de instruções seja composto de "n" instruções? Basta então fazer uso das chaves para indicar aonde inicia e aonde termina o bloco de instruções do comando **if** (ASCENCIO; CAMPOS, 2002; ALBANO; ALBANO, 2010; BACKES, 2013; PAES, 2016). Veja um exemplo em forma de exercício a seguir.

### 2.2.2 Exercício de Exemplo

Faça um programa em C que receba um número inteiro e verifique se este número é maior que 20, em caso afirmativo o programa deverá multiplicar o valor por 2 e após o cálculo imprimir a mensagem: "Resultado: <valor do resultado>", em que <valor do resultado> deve ser substituído pelo resultado do cálculo.

Fonte: Adaptado de Lopes e Garcia (2002, p. 79)

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 void main() {
4     int num, resul;
5     printf("Informe o numero:");
6     scanf("%d", &num);
7
8     if (num > 20) {
9         resul = num * 2;
10        printf("Resultado: %d", resul);
11    }
12 }
```

Neste caso, o programa deve executar duas instruções caso o resultado da expressão condicional seja verdadeiro. A primeira instrução será a multiplicação do valor informado pelo usuário por 2, e a segunda instrução será impressão do resultado na tela. Assim, a resposta do exercício nos permite analisar um exemplo de execução de mais de uma instrução no comando **if**. Neste caso, foi necessário declarar uma variável adicional para armazenar o resultado do cálculo, que foi denominada como **resul**. Exceto este ponto, a resposta segue igual à resposta do exercício anterior até a linha 7.

Na linha 8 ocorreu a primeira diferença, no final da linha, note a presença da chave, "{". Esta chave indica o início do bloco de instruções do **if**, a partir deste ponto, pode-se colocar "n" instruções até a indicação de término do bloco de instruções, que no exemplo é representada pelo fechamento da chave, "}" na linha 11. Todas as n instruções que estiverem compreendidas entre estas duas chaves irão ser processadas caso o resultado da expressão de condição seja verdadeiro.

Mas, o que o programa fará se o resultado da expressão não for verdadeiro? Ou seja, o que ele fará se o valor informado pelo usuário foi menor ou igual a 20? Bem, no exemplo apresentado, o programa não fará nada! Pois, como não foi solicitado no enunciado do exercício anterior, não foi indicado o que o programa deveria fazer neste caso. A seguir, o exercício foi aprimorado para exemplificar o uso da cláusula **else**.

### 2.2.3 Exercício de Exemplo

Faça um programa em C que receba um número inteiro e verifique se este número é maior que 20, em caso afirmativo o programa deverá multiplicar o valor por 2 e em caso negativo deve multiplicar por 4. Após realizar o cálculo o programa deve imprimir a mensagem: "Resultado: <valor do resultado>", em que <valor do resultado> deve ser substituído pelo resultado do cálculo.

Fonte: Adaptado de [Lopes e Garcia \(2002, p. 79\)](#)

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 void main() {
4     int num, resul;
5     printf("Informe o numero:");
6     scanf("%d", &num);
7
8     if (num > 20) {
9         resul = num * 2;
10        printf("Resultado: %d", resul);
11    }
12    else {
```

```
13     resul = num * 4;
14     printf("Resultado: %d", resul);
15 }
16 }
```

Na resposta do exercício apresentada, o programa teve que executar a instrução de multiplicação por 2, caso o usuário informasse o valor seja superior à 20 e a instrução de multiplicação por 4 caso o valor seja inferior ou igual a 20. Veja que neste caso, um valor igual a 20 foi considerado pela cláusula **else**, isso porque a expressão na condição indica **num > 20**, desta forma, qualquer valor que não seja maior que 20 será processado pela cláusula **else**.

Um ponto chama a atenção na resposta apresentada, ao observar as linhas 10 e 14, chega-se à conclusão de que são iguais, ou seja, embora seja necessário realizar a impressão do resultado nos dois casos, como a variável utilizada e o texto a ser impresso são os mesmos, independente do resultado da condição, então o comando permaneceu inalterado, neste caso, o ideal é que essa instrução seja posicionada fora da estrutura de decisão **if-else** para evitar redundância de código. Assim, poderia também remover as chaves na estrutura de decisão, pois passa a ter apenas uma instrução em cada fluxo e, portanto, não seriam mais necessárias as chaves, contudo, a presença delas não influi negativamente, então elas serão mantidas. Veja a seguir o ajuste feito na resposta.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 void main() {
4     int num, resul;
5     printf("Informe o numero:");
6     scanf("%d", &num);
7
8     if (num > 20) {
9         resul = num * 2;
10    }
11    else {
12        resul = num * 4;
13    }
14    printf("Resultado: %d", resul);
15 }
```

Agora note que neste caso, o comando **printf()** foi movido para fora do bloco de decisão, pois como ela não se altera em relação à condição testada, então esta é comum às duas situações e pode ser movida para fora deste fluxo de decisão.

## 2.3 Indentação

Nos três exercícios de exemplo, apresentados até agora nesta aula, há indentação. Apesar de não ser obrigatório o seu uso em linguagem C, é muito comum, pois torna o código-fonte mais legível. Se imagine tendo que percorrer centenas de linhas de código em um arquivo e todas as linhas de código estão encostadas na margem? A leitura se torna mais lenta e pesada, quando usada a indentação para expressar a relação de dependência é bem mais fácil identificar, por exemplo, quais instruções serão executadas se uma condição em um **if** for verdadeira ou falsa (EDELWEISS; LIVI, 2014).

Assim, é comum incluir espaços ou tabulações em instruções que estão dentro de um determinado bloco, como é o caso desta aula, os blocos pertencentes à uma

cláusula **if** ou **else**. Desta forma, sempre que utilizar a indentação daqui em diante, lembre-se que a está utilizando para expressar a relação de dependência que há entre um bloco de instruções com outro. Veja na figura 10 a diferença de escrita.

<pre>if (num != 0)     printf("%d e zero!", num); else     printf("%d e diferente de zero", num);</pre>	<pre>if (num != 0) printf("%d e zero!", num); else printf("%d e diferente de zero", num);</pre>
Com indentação	Sem indentação

Figura 10 – Código com indentação e sem indentação

Apesar do código na figura 10 apresentar somente 4 linhas, fica claro como o código com indentação é mais legível que o código sem indentação. No código sem indentação não é tão claro qual das linhas irá ou não executar, ou mesmo se há algum fluxo. Como, no exemplo são apenas 4 linhas, talvez não seja tão difícil identificar os fluxos ou instruções dentro dos fluxos, mas imagine se no bloco do **if** houvessem 30 instruções e outras 30 no bloco do **else**, neste caso, quando você fosse navegar por este código, a cláusula **else** poderia até passar despercebida.

## 2.4 Cláusula if-else com n blocos de instruções

Até o momento foram apresentadas situações que envolvam a execução ou não de um fluxo, neste caso, a presença apenas do comando **if** e a execução de um fluxo ou outro, nesta situação, a presença do **if-else**, mas e se for necessário que o programa decida entre 3 fluxos diferentes? Ou ainda, 4 fluxos ou mais? Quando precisar tratar "n" fluxos na decisão das instruções a serem executadas, associe o uso do **else** com o **if**. Novamente, veja um exercício para exemplificar essa situação.

### 2.4.1 Exercício de Exemplo

O escritório de contabilidade Super Contábil está realizando o reajuste do salário dos funcionários de todos os seus clientes. Para isso, estão utilizando como referência o reajuste acordado com os sindicatos de cada classe trabalhadora, conforme apresentado na tabela a seguir. Para ajudar o escritório nesta tarefa, faça um programa em C que receba o salário atual, o cargo conforme especificado na tabela a seguir e realize o cálculo do reajuste do salário. Ao término do cálculo o programa deve imprimir o valor do reajuste e o novo salário do funcionário.

<i>Cód. cargo</i>	<i>Cargo</i>	<i>% reajuste acordado</i>
1	Auxiliar de escritório	7%
2	Secretário(a)	9%
3	Cozinheiro(a)	5%
4	Entregador(a)	12%

Para resolver este exercício foi utilizado o código do cargo para determinar a qual cargo pertence o funcionário, no qual o salário está sendo reajustado, assim, foi declarada uma variável do tipo inteiro para armazenar o cargo e foram declaradas duas variáveis do tipo real para armazenar o salário atual do funcionário e o valor do reajuste. Além disso, foi utilizada a estrutura de decisão **if-else** para decidir qual fluxo executar de

acordo com o cargo do funcionário. Como são 4 cargos, então são necessários 4 fluxos distintos na estrutura de decisão. Veja a seguir a resposta do exercício e os comentários.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 void main() {
4     int cargo;
5     float salAtual, reajuste;
6     printf("Informe o cargo do funcionario:");
7     scanf("%d", &cargo);
8     printf("Informe o salario atual:");
9     scanf("%f", &salAtual);
10
11     if (cargo == 1)
12         reajuste = (salAtual * 7) / 100;
13     else if (cargo == 2)
14         reajuste = (salAtual * 9) / 100;
15     else if (cargo == 3)
16         reajuste = (salAtual * 5) / 100;
17     else
18         reajuste = (salAtual * 12) / 100;
19     printf("O reajuste e: %f", reajuste);
20     printf("O novo salario e: %f", salAtual + reajuste);
21 }
```

A resposta apresentada para o exercício traz uma possível solução para o exemplo. Apesar de mencionar pela primeira vez as palavras "possível solução", o mesmo conceito se aplica aos exercícios anteriores e aos próximos, quero dizer com isso, que não há uma única solução para os exercícios apresentadas, nem mesmo os mais simples exercitados, todos eles possuem soluções diferentes das apresentadas que serão satisfatórias. E dificilmente dois programadores irão produzir soluções iguais para um mesmo problema, é provável que produzam soluções parecidas, mas é bem pouco provável que façam uma resposta igual, mesmo que seja apenas a diferença nos nomes propostos para as variáveis, sempre haverá alguma diferença pelo simples fato de que as pessoas pensam diferente umas das outras e a programação permite construções distintas para um mesmo problema.

Bem, em relação aos comentários da solução proposta, veja que nas linhas 4 e 5 foram declaradas as variáveis que são necessárias no programa, uma variável do tipo inteiro (int) para armazenar o cargo do funcionário, denominada por **cargo**, duas variáveis do tipo real (float) para armazenar o salário atual, **salAtual** e o resultado do cálculo do reajuste a ser aplicado ao salário atual, denominada **reajuste**. Entre as linhas do programa, 6 a 9, foram utilizadas as funções **printf()** e **scanf()** para solicitar ao usuário que informe o cargo do funcionário e o salário atual. O cargo será utilizado no cálculo do reajuste e o salário atual também.

Na linha 11 do exercício de exemplo é iniciado o fluxo de decisão que considera o **cargo** do funcionário como sendo o fator de decisão para a escolha entre os fluxos, pois é o cargo que determina o reajuste a ser aplicado. Desta forma, a linha 11 traz a seguinte condição: "**cargo == 1**", note que o teste de igualdade é determinado pelo uso da igualdade dupla, conforme pode ser visto na tabela 8, assim se o usuário informar o código "1" para o cargo, então o programa executará o bloco de instruções do primeiro fluxo, ou seja, a linha 12. Veja como foi feito o cálculo de percentual de reajuste: "**reajuste = (salAtual \* 7) / 100;**", nota-se que o salário atual foi multiplicado por 7 conforme o cargo 1 e o resultado dessa multiplicação foi dividido por 100 para então, o resultado

desta expressão ser atribuído à variável **reajuste**. A presença dos parênteses "(" na primeira parte da expressão matemática, garante que esta será sempre a primeira a ser realizada, assim, caso a operação fosse a soma, por exemplo, ela seria executada primeiro que a segunda parte da expressão, mesmo a segunda parte sendo uma divisão.

Na linha 13, foi incluído o segundo fluxo de decisão e a primeira ocorrência da combinação entre as instruções **else** e **if** para possibilitar que uma nova condição seja testada caso a anterior seja falsa. Esse é o princípio do funcionamento da combinação das duas instruções, se a primeira retornar falso como resultado, ao invés de apenas executar o próximo bloco, será feito um novo teste de condição, se o próximo teste também retornar falso, será realizado um novo teste no próximo fluxo encontrado e assim por diante, até o último fluxo, em que há apenas a cláusula **else**.

As linhas 17 e 18 trazem o último fluxo, totalizando assim os 4 fluxos necessários para o teste dos cargos. As linhas 19 e 20 possibilitam a conclusão do exercício pela impressão do reajuste a ser aplicado, na linha 19, e do novo salário, na linha 20. Note na linha 20, que por opção, o cálculo do novo salário, ou seja, "**salAtual + reajuste**", foi realizado no comando **printf()**, neste caso, o valor do salário novo não foi armazenado em nenhuma variável, foi apenas calculado para impressão.

## 2.5 Cláusula if-else com condições compostas

Até agora, foi ensinado a utilizar a estrutura de decisão **if-else** com apenas um bloco, com dois blocos, e com **n** blocos de instruções. Foi ensinado também a utilizar os operadores relacionais apresentados na tabela 8. Agora é necessário aprender a fazer testes de condição com a combinação de operadores lógicos, em que é possível fazer uso de conjunções, disjunções e negação (LOPES; GARCIA, 2002). Em linguagem C é possível combinar vários operadores em uma mesma condição, inclusive pode-se também combinar operadores diferentes. Para iniciar, veja primeiro a tabela 9 que apresenta os operadores lógicos. Na sequência veja um exercício para exemplificar o uso da combinação de condições.

Tabela 9 – Operadores lógicos

Operador Lógico	Representação em C	Exemplo
E (conjunção)	&&	$x > 1 \ \&\& \ x < 19$
OU (disjunção)	(duas barras verticais)	$x == 1 \    \ x == 2$
NÃO (negação)	! (exclamação)	! Continuar

Fonte: Adaptado de Damas (2007, p. 63)

### 2.5.1 Exercício de Exemplo

O hospital local está fazendo uma campanha para receber doação de sangue. O pro-penso doador deve inicialmente se cadastrar informando o seu nome completo, sua idade, seu peso, responder a um breve questionário e apresentar um documento oficial com foto. Faça um programa que permita ao hospital realizar o cadastro dos voluntários para avaliar a aptidão quanto à doação de sangue. Para estar apto a doar, o voluntário deve ter idade entre 16 e 69, pesar pelo menos 50 kg, estar bem alimentado e não estar resfriado. O programa deve ler os dados e imprimir no final o nome do voluntário e se ele está apto ou não.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 void main() {
4     char nome[30];
5     int idade, bemAlimentado, resfriado;
6     float peso;
7     printf("Informe o nome:");
8     scanf("%s", &nome);
9     printf("Informe o peso:");
10    scanf("%f", &peso);
11    printf("Informe a idade:");
12    scanf("%d", &idade);
13    printf("Esta bem alimentado? <1-SIM / 0-NAO>");
14    scanf("%d", &bemAlimentado);
15    printf("Esta resfriado? <1-SIM / 0-NAO>");
16    scanf("%d", &resfriado);
17    if (peso >= 50 && (idade >= 16 && idade <= 69)
18        && bemAlimentado && !resfriado)
19        printf("O voluntario %s esta apto!", nome);
20    else
21        printf("O voluntario %s NAO esta apto", nome);
22 }
```

Pode-se ver na resposta sugerida para o exercício, que na linha 4 foi feita a declaração da variável que será responsável por armazenar o nome do voluntário. Veja que o nome da variável está acompanhado do trecho "[30]", isso indica que foi declarado um **char** com tamanho 30, é comum fazer esse tipo de declaração quando é necessário armazenar texto com tamanho superior a 1. Nas linhas 5 e 6 foram declaradas as demais variáveis para armazenar o peso, a idade e os indicativos de bem alimentado e resfriado. No caso dos indicativos, note que as variáveis foram definidas com o tipo **int**, desta forma, adotou-se os valores 1 quando afirmativo e 0 quando negativo.

Entre as linhas 7 e 16 da resposta proposta para o exercício, foram incluídas as instruções para que o programa leia as informações fornecidas pelo usuário. Na linha 17 inicia-se o bloco de decisão. Note que o comando **if** inclui várias condições que são combinadas pelo operador lógico **"&&"** que corresponde ao "E (conjunção) conforme a tabela 9. Na figura 11 pode ser visto em detalhes qual trecho do código corresponde à cada condição presente na estrutura de decisão **if**. Veja que a instrução inclui 4 condições, sendo que a segunda condição é composta e por sua vez possui 2 condições, totalizando assim 5 condições.

A figura 11 mostra também os operadores lógicos, sendo o operador **"&&"** que é responsável por combinar as condições. O operador de conjunção indica que, para que essa expressão de condição seja considerada verdadeira, todas as condições devem ser verdadeiras, basta que uma delas seja falsa para que a combinação de todas as condições seja considerada falsa. Exemplo, se a idade informada pelo usuário for "15", toda a expressão de condição será considerada falsa, pois todas devem ser verdadeiras para que o conjunto de condições seja considerado verdadeiro. E também o operador de negação (**"!"**), que neste caso, é utilizado para retornar o contrário do que está armazenado na variável **resfriado**, assim, se o usuário informou, por exemplo, "1" indicando que o voluntário está resfriado, então a negação irá retornar o contrário, "0" (zero), e no teste da condição a expressão será considerada falsa, pois não é aceitável um doador resfriado.

Então, nos testes lógicos em linguagem C, sempre será **falso** quando o valor inteiro

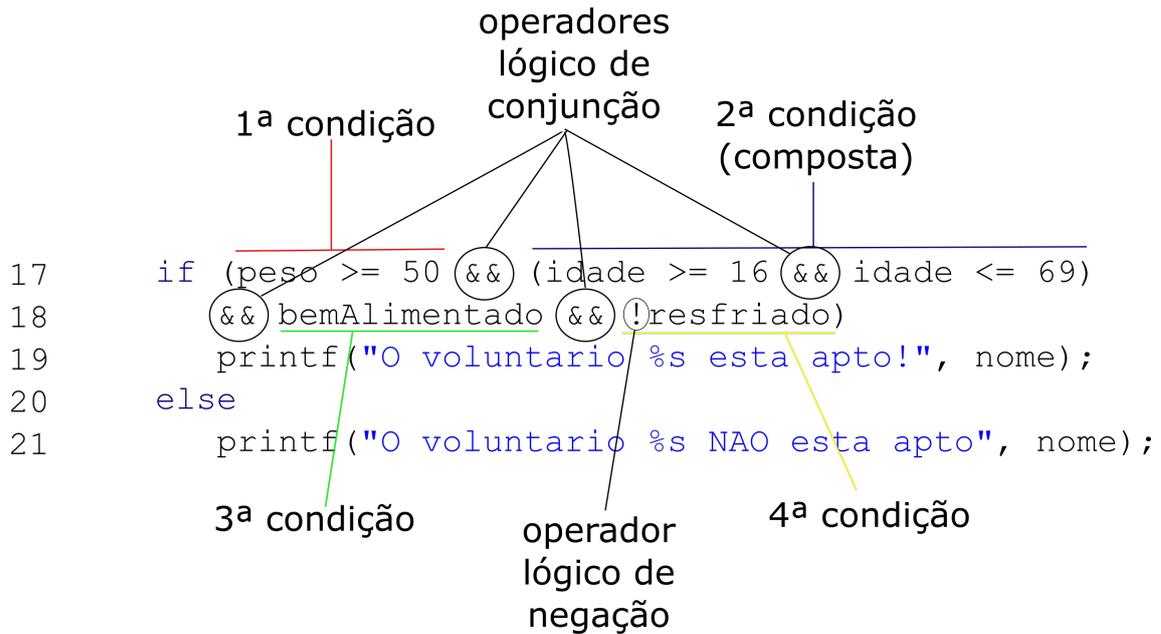


Figura 11 – Exemplo de aplicação dos operadores lógicos

for igual à "0" (zero) e será **verdadeiro** quando o valor for diferente de zero, ou seja, qualquer outro que não seja zero, incluindo números negativos (SCHILDT, 1996, p. 62). Desta forma, no exercício de exemplo apresentado, é preciso uma pessoa bem alimentada, assim, se o usuário responder "0" o teste da condição irá considerar falso, mas se o usuário responder qualquer número diferente de zero, então será considerado verdadeiro, isso quer dizer, que apesar do programa orientar ao usuário responder "1-SIM / 0-NAO" conforme a linha 15 da solução proposta, se por algum motivo o usuário responder, por exemplo, "2", será considerado verdadeiro, mesmo que ele tenha respondido um valor fora da faixa disponibilizada pelo programa. Neste caso, o ideal seria forçar o usuário a responder apenas conforme indicado pelo programa, mas isso é assunto para a próxima aula.

Veja agora mais um exercício de exemplo, neste caso, utilizando o operador lógico "OU" (disjunção).

## 2.5.2 Exercício de Exemplo

Segundo uma tabela médica, o peso ideal está relacionado com a altura e o sexo. Faça um programa em C que receba a altura e o sexo de uma pessoa, após isso calcule e imprima o seu peso ideal, utilizando as seguintes fórmulas:

- Para homens:  $(72,7 * A) - 58$
- Para mulheres:  $(62,1 * A) - 44,7$
- Em que:

A = Altura

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 void main() {
4     int altura;
5     char sexo;
6     float pesoIdeal;
7     printf("Informe a altura:");
8     scanf("%d", &altura);
9     printf("Informe o sexo: <M ou F>");
10    scanf("%c", &sexo);
11    if (sexo == 'm' || sexo == 'M')
12        pesoIdeal = (72.7 * altura) - 58;
13    else
14        pesoIdeal = (62.1 * altura) - 44.7;
15    printf("O peso ideal e: %f", pesoIdeal);
16 }
```

O objetivo deste exemplo é mostrar o uso do operador lógico de disjunção, o "OU". Entre as linhas 1 e 10 foi feita a declaração das variáveis e a leitura dos valores informados pelo usuário, é importante notar que a variável **sexo** foi declarada como **char**, ou seja, para possibilitar ao usuário que informe uma letra. As letras tem variações entre minúscula e maiúscula, neste exemplo, serão utilizadas duas consoantes, "M" ou "m" e "F" ou "f". Desta forma, ao informar o sexo, pode ocorrer do usuário informar com letra maiúscula ou minúscula mesmo com a orientação da linha 9.

A estrutura de decisão será responsável então, por tratar essa variação entre letras minúsculas ou maiúsculas. Na linha 11 da resposta proposta são apresentadas então as duas condições para a validade da condição geral, ou seja, se o usuário informar "M" ou "m" a condição geral será verdadeira e a instrução para o cálculo do peso ideal será feito para o homem na linha 12. Se o usuário informar qualquer outra letra, então a condição será considerada falsa e será executado o cálculo para o peso ideal da mulher na linha 14. Após realizar o cálculo, o resultado do peso ideal é impresso pela instrução da linha 16 da resposta do exercício.

## 2.6 Cláusula if-else com condições aninhadas

A estrutura de decisão **if-else** permite também o uso de condições aninhadas (encaixadas) que são úteis quando é necessário tomar uma decisão dentro de outra. A seguir um exercício para exemplificar o uso de **if** aninhado.

### 2.6.1 Exercício de Exemplo

Faça um programa em C que leia o destino do passageiro, se a viagem inclui retorno (ida e volta) e informe o preço da passagem conforme a tabela a seguir:

CÓD. DESTINO	DESTINO	IDA	IDA E VOLTA
1	Região Norte	500,00	900,00
2	Região Nordeste	350,00	650,00
3	Região Centro-oeste	350,00	600,00
4	Região Sul	300,00	550,00

Fonte: Adaptado de [Lopes e Garcia \(2002, p. 113\)](#)

Para resolver este exercício será necessário primeiro verificar qual é o destino, depois de verificar o destino, verificar se o trecho inclui somente ida ou ida e volta, ou seja, há uma verificação condicionada a outra, a primeira será o destino e a segunda e aninhada à primeira, será a condição do trecho.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 void main() {
4     int destino, trecho;
5     printf("Informe o destino conforme tabela a seguir: \n");
6     printf("1-Regiao Norte \t 3-Regiao Centro-oeste \n");
7     printf("2-Regiao Nordeste \t 4-Regiao Sul \n");
8     scanf("%d", &destino);
9     printf("Informe o trecho: <1-IDA ou 2-IDA E VOLTA>");
10    scanf("%d", &trecho);
11    if (destino == 1) {
12        if (trecho == 1)
13            printf("Regiao norte[IDA] = 500,00");
14        else
15            printf("Regiao norte[IDA E VOLTA] = 900,00");
16    }
17    else if (destino == 2) {
18        if (trecho == 1)
19            printf("Regiao nordeste[IDA] = 350,00");
20        else
21            printf("Regiao nordeste[IDA E VOLTA] = 650,00");
22    }
23    else if (destino == 3) {
24        if (trecho == 1)
25            printf("Regiao centro-oeste[IDA] = 350,00");
26        else
27            printf("Regiao centro-oeste[IDA E VOLTA] = 600,00");
28    }
29    else {
30        if (trecho == 1)
31            printf("Regiao sul[IDA] = 300,00");
32        else
33            printf("Regiao sul[IDA E VOLTA] = 550,00");
34    }
35 }
```

Até a linha 10 da resposta apresentada para o exercício, há instruções que permitem ao usuário informar o destino e o trecho conforme a tabela do exercício. A partir da linha 11 foi incluída a estrutura de condição com **if** aninhado. Note que, primeiro é feita a comparação com o **destino** na linha 11, pois o **trecho** irá apresentar diferentes valores para cada diferente destino, por isso, é necessário analisar a condição do trecho "dentro" da condição do destino. Na primeira condição, se o destino escolhido é igual à "1", ou seja, "Região Norte", então a instrução na linha 12 será executada, se o trecho escolhido também é igual à "1", então a instrução na linha 13 será executada, mas se o trecho escolhido é diferente de "1", então a instrução na linha 15 será executada.

As linhas seguintes seguem o mesmo padrão, mas atendendo aos demais destinos e trechos da tabela do exercício, a linha 29 traz a última condição para destino, que é contrária aos destinos 1, 2 e 3, ou seja, qualquer outro destino diferente destes levarão à execução das instruções a partir da linha 29. Não há um limite para incluir um

**if** aninhado, "dentro", de outro **if** em níveis, é possível incluir quantos níveis forem necessários, contudo, um número de níveis muito grande não é muito prático em termos de programação, principalmente no que diz respeito à manutenção do código-fonte, então é melhor evitar muitos níveis.

## 2.7 Cláusula switch

O **switch** é um comando com possibilidades mais simplificadas que o **if-else**, desta forma, permite apenas a comparação de igualdade com variáveis do tipo **int**, **char** e **long**. O **switch** é vantajoso quando é necessário fazer muitas comparações, pois neste caso, o **switch** irá oferecer maior agilidade na implementação (LAUREANO, 2005; MIZRAHI, 2008). Veja a seguir a sintaxe:

```
1 //Sintaxe:
2 switch (expressao) {
3     case constante1:
4         instrucoes1;
5         break;
6     case constante2:
7         instrucoes2;
8         break;
9     ...
10    default:
11        instrucoes;
12 }
```

Para utilizar o **switch** basta substituir na linha 2 da sintaxe, a palavra **expressao** pelo nome da variável que terá sua expressão avaliada, na linha 3 substituir a palavra **constante1** pela constante a ser comparada com o conteúdo da variável em expressao, na linha 4 substituir **instrucoes1** pelas instruções que pretende-se executar caso a comparação seja verdadeira. A mesma substituição deve ser feita nas linhas 6 e 7 e novas instruções **case** podem ser adicionadas, tantas quantas forem necessárias. A cláusula **default** funciona como o último **else** em um conjunto de instruções **if**, ou seja, se nenhuma condição anterior é verdadeira, então as instruções em **default** serão executadas.

A cláusula **break** é responsável pela parada na execução das instruções, pois caso não seja colocada, as validações presentes no **switch** continuarão a ser executadas, mesmo que um **case** já tenha resultado em verdadeiro. A cláusula **default** é opcional. A seguir um exercício de exemplo.

### 2.7.1 Exercício de Exemplo

Escreva um programa em linguagem C que leia um peso na Terra e o número de um planeta e imprima o valor correspondente do peso neste planeta. A relação de planetas é dada a seguir juntamente com o valor das gravidades relativas à Terra.

Código	Gravidade Relativa	Planeta
1	0,37	Mercúrio
2	0,88	Vênus
3	0,38	Marte
4	2,64	Júpiter
5	1,15	Saturno
6	1,17	Urano

Para calcular o peso no planeta use a fórmula:

$$PP = \frac{PT}{10} \times G \quad (2.1)$$

Em que:

- PP = Peso no planeta
- PT = Peso na Terra
- G = Gravidade relativa

Fonte: Adaptado de [Lopes e Garcia \(2002, p. 77\)](#)

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     //declaracao das variaveis
7     float pesoTerra, pesoPlaneta;
8     int numPlaneta;
9     //leitura do peso na terra e escolha do planeta
10    printf("Informe o peso na terra:\n");
11    scanf("%f", &pesoTerra);
12    printf("Informe o numero do planeta conforme a tabela: \n");
13    printf("1-Mercurio\t 2-Venus\t 3-Marte\n");
14    printf("4-Jupiter\t 5-Saturno\t 6-Urano\n");
15    scanf("%d", &numPlaneta);
16
17    //switch responsavel pela escolha do calculo
18    switch (numPlaneta) {
19        case 1:
20            pesoPlaneta = (pesoTerra / 10) * 0.37;
21            break;
22        case 2:
23            pesoPlaneta = (pesoTerra / 10) * 0.88;
24            break;
25        case 3:
26            pesoPlaneta = (pesoTerra / 10) * 0.38;
27            break;
28        case 4:
29            pesoPlaneta = (pesoTerra / 10) * 2.64;
30            break;
31        case 5:
32            pesoPlaneta = (pesoTerra / 10) * 1.15;
33            break;

```

```
34     case 6:
35         pesoPlaneta = (pesoTerra / 10) * 1.17;
36         break;
37     default:
38         printf("Planeta invalido!\n");
39     }
40     //impressao do peso no planeta escolhido
41     printf("Peso no planeta escolhido: %f \n", pesoPlaneta);
42 }
```

Como pode ser visto na resposta proposta para o exercício de exemplo, a partir da linha 18, foi incluído o uso da cláusula **switch**, e foi utilizada como expressão de teste de fluxo, a variável **numPlaneta**, pois conforme solicitado no exercício, o peso deve ser convertido conforme o planeta escolhido, e esta variável é utilizada no programa para armazenar o código do planeta escolhido, assim, se o usuário escolher o código 1, referente ao planeta Mercúrio, a instrução na linha 19, **case 1:**, irá retornar verdadeiro após avaliação e conseqüentemente o programa irá executar a linha 20, responsável pelo cálculo do peso e a linha 21 responsável pela saída do comando **switch**. De forma similar, o programa irá se comportar para cada código de planeta escolhido pelo o usuário, caso o usuário informe um código que não está na faixa estipulada, então a instrução na linha 37, **default:** irá retornar verdadeiro e conseqüentemente a linha 38 será executada, apresentando assim uma mensagem informativa para o usuário.

Caso o programa apresentado não tivesse os comandos **break** em cada comando **case**, então, ao avaliar um comando **case**, o programa iria continuar a avaliar os próximos comandos **case** encontrados na sequência, mesmo que um deles retorne verdadeiro. Assim o uso da cláusula **break** permite interromper o **switch** caso o programa já tenha identificado a condição esperada.

## 2.8 Resumo da Aula

Na aula 2 foram apresentados os conceitos necessários para construir um programa em linguagem C com fluxos de execução distintos, ou seja, aplicações que executem tarefas ou ações diferentes de acordo com os dados de entrada do programa. Assim, o programa pode por exemplo, decidir efetuar um cálculo em detrimento de outro em função dos dados de entrada, em computação, isso caracteriza uma estrutura de decisão.

Neste sentido, foram apresentadas duas cláusulas de estrutura de decisão em linguagem C, a cláusula **if-else** e **switch**. A cláusula **if-else** é a mais poderosa e mais utilizada, pois dá várias possibilidades ao programador, como estabelecer se um bloco de instruções será ou não executado, ou definir dois blocos de instruções, sendo um dos dois executados sempre, ou ainda, estabelecer **n** blocos de instruções. Além disso, a cláusula **if** permite avaliar expressões de igualdade, de diferença, entre outras. Este comando possibilita também combinar expressões de relação diferentes, por exemplo, combinar uma expressão de igualdade com uma expressão de diferença.

A cláusula **switch** também possibilita situações variadas, mas tem algumas limitações em detrimento do **if**, por exemplo, enquanto o **if** possibilita expressões com operadores relacionais variados, o **switch** só permite a avaliação de igualdade. No entanto, o **switch** também tem vantagens em relação ao **if**, por exemplo, quando há um problema que requer a avaliação de várias comparações de igualdade, em geral, é mais produtivo programar este problema com **switch** do que com o **if**. Desta forma, é possível concluir que, não há porque dizer que o **if** é melhor que o **switch** ou vice-versa, mas que ambas tem suas aplicações.

## 2.9 Exercícios da Aula

Os exercícios desta lista foram Adaptados de [Lopes e Garcia \(2002, p. 79-120\)](#).

1. Faça um programa em C que leia dois valores numéricos inteiros e efetue a adição, caso o resultado seja maior que 10, apresentá-lo.
2. Faça um programa em C que leia dois valores inteiros e efetue a adição. Caso o valor somado seja maior que 20, este deverá ser apresentado somando-se a ele mais 8, caso o valor somado seja menor ou igual a 20, este deverá ser apresentado subtraindo-se 5.
3. Faça um programa que leia um número e imprima uma das duas mensagens: "É múltiplo de 3" ou "Não é múltiplo de 3".
4. Faça um programa que leia um número e informe se ele é ou não divisível por 5.
5. Faça um programa em C que leia um número e informe se ele é divisível por 3 e por 7.
6. A prefeitura do Rio de Janeiro abriu uma linha de crédito para os funcionários estatutários. O valor máximo da prestação não poderá ultrapassar 30% do salário bruto. Faça um programa em linguagem C que permita entrar com o salário bruto e o valor da prestação e informar se o empréstimo pode ou não ser concedido.
7. Faça um programa em C que leia um número e indique se o número está compreendido entre 20 e 50 ou não.
8. Faça um programa que leia um número e imprima uma das mensagens: "Maior do que 20", "Igual a 20" ou "Menor do que 20".
9. Faça um programa em C que permita entrar com o ano de nascimento da pessoa e com o ano atual. O programa deve imprimir a idade da pessoa. Não se esqueça de verificar se o ano de nascimento informado é válido.
10. Faça um programa em C que leia três números inteiros e imprima os três em ordem crescente.
11. Faça um programa que leia 3 números e imprima o maior deles.
12. Faça um programa que leia a idade de uma pessoa e informe:
  - Se é maior de idade
  - Se é menor de idade
  - Se é maior de 65 anos
13. Faça um programa que permita entrar com o nome, a nota da prova 1 e a nota da prova 2 de um aluno. O programa deve imprimir o nome, a nota da prova 1, a nota da prova 2, a média das notas e uma das mensagens: "Aprovado", "Reprovado" ou "em Prova Final" (a média é 7 para aprovação, menor que 3 para reprovação e as demais em prova final).
14. Faça um programa que permita entrar com o salário de uma pessoa e imprima o desconto do INSS segundo a tabela seguir:

Salário	Faixa de Desconto
Menor ou igual à R\$600,00	Isento
Maior que R\$600,00 e menor ou igual a R\$1200,00	20%
Maior que R\$1200,00 e menor ou igual a R\$2000,00	25%
Maior que R\$2000,00	30%

15. Um comerciante comprou um produto e quer vendê-lo com um lucro de 45% se o valor da compra for menor que R\$20,00, caso contrário, o lucro será de 30%. Faça um programa em C que leia o valor do produto e imprima o valor da venda.
16. A confederação brasileira de natação irá promover eliminatórias para o próximo mundial. Faça um programa em C que receba a idade de um nadador e imprima a sua categoria segundo a tabela a seguir:

Categoria	Idade
Infantil A	5 - 7 anos
Infantil B	8 - 10 anos
Juvenil A	11 - 13 anos
Juvenil B	14 - 17 anos
Sênior	maiores de 18 anos

17. Depois da liberação do governo para as mensalidades dos planos de saúde, as pessoas começaram a fazer pesquisas para descobrir um bom plano, não muito caro. Um vendedor de um plano de saúde apresentou a tabela a seguir. Faça um programa que entre com o nome e a idade de uma pessoa e imprima o nome e o valor que ela deverá pagar.

Idade	Valor
Até 10 anos	R\$30,00
Acima de 10 até 29 anos	R\$60,00
Acima de 29 até 45 anos	R\$120,00
Acima de 45 até 59 anos	R\$150,00
Acima de 59 até 65 anos	R\$250,00
Maior que 65 anos	R\$400,00

18. Faça um programa que leia um número inteiro entre 1 e 12 e escreva o mês correspondente. Caso o usuário digite um número fora desse intervalo, deverá aparecer uma mensagem informando que não existe mês com este número. Utilize o **switch** para este problema.
19. Em um campeonato nacional de arco-e-flecha, tem-se equipes de três jogadores para cada estado. Sabendo-se que os arqueiros de uma equipe não obtiveram o mesmo número de pontos, criar um programa em C que informe se uma equipe foi classificada, de acordo com a seguinte especificação:
- Ler os pontos obtidos por cada jogador da equipe;
  - Mostrar esses valores em ordem decrescente;
  - Se a soma dos pontos for maior do que 100, imprimir a média aritmética entre eles, caso contrário, imprimir a mensagem "Equipe desclassificada".

20. O banco XXX concederá um crédito especial com juros de 2% aos seus clientes de acordo com o saldo médio no último ano. Faça um programa que leia o saldo médio de um cliente e calcule o valor do crédito de acordo com a tabela a seguir. O programa deve imprimir uma mensagem informando o saldo médio e o valor de crédito.

Saldo Médio	Percentual
de 0 a 500	nenhum crédito
de 501 a 1000	30% do valor do saldo médio
de 1001 a 3000	40% do valor do saldo médio
acima de 3001	50% do valor do saldo médio

21. A biblioteca de uma Universidade deseja fazer um programa que leia o nome do livro que será emprestado, o tipo de usuário (professor ou aluno) e possa imprimir um recibo conforme mostrado a seguir. Considerar que o professor tem dez dias para devolver o livro e o aluno só três dias.
- Nome do livro:
  - Tipo de usuário:
  - Total de dias:
22. Construa um programa que leia o percurso em quilômetros, o tipo do carro e informe o consumo estimado de combustível, sabendo-se que um carro tipo C faz 12 km com um litro de gasolina, um tipo B faz 9 km e o tipo C, 8 km por litro.
23. Crie um programa que informe a quantidade total de calorias de uma refeição a partir da escolha do usuário que deverá informar o prato, a sobremesa, e bebida conforme a tabela a seguir.

Prato	Sobremesa	Bebida
Vegetariano 180cal	Abacaxi 75cal	Chá 20cal
Peixe 230cal	Sorvete diet 110cal	Suco de laranja 70cal
Frango 250cal	Mousse diet 170cal	Suco de melão 100cal
Carne 350cal	Mousse chocolate 200cal	Refrigerante diet 65cal

24. A polícia rodoviária resolveu fazer cumprir a lei e vistoriar veículos para cobrar dos motoristas o DUT. Sabendo-se que o mês em que o emplacamento do carro deve ser renovado é determinado pelo último número da placa do mesmo, faça um programa que, a partir da leitura da placa do carro, informe o mês em que o emplacamento deve ser renovado.
25. A prefeitura contratou uma firma especializada para manter os níveis de poluição considerados ideais para um país do 1º mundo. As indústrias, maiores responsáveis pela poluição, foram classificadas em três grupos. Sabendo-se que a escala utilizada varia de 0,05 e que o índice de poluição aceitável é até 0,25, fazer um programa que possa imprimir intimações de acordo com o índice e a tabela a seguir:

Índice	Indústrias que receberão intimação
0,3	1º grupo
0,4	1º e 2º grupos
0,5	1º, 2º e 3º grupos



# Estruturas de Iteração

## Metas da Aula

1. Entender e praticar os conceitos da sintaxe utilizada na linguagem C para estruturas de iteração.
2. Aplicar variadas situações relacionadas ao fluxo de iteração em programação com o objetivo de cobrir diversificadas possibilidades vivenciadas na programação cotidiana.
3. Escrever programas que farão uso de estruturas de iteração.

## Ao término desta aula, você será capaz de:

1. Escrever programas em linguagem C que sejam capazes de resolver problemas que envolvam, por exemplo, a repetição de várias instruções em função de uma determinada condição.
2. Escrever programas que manipulem informações repetidas vezes considerando variadas situações e condições.

## 3.1 Estruturas de Iteração

Até esta parte do livro foram trabalhados conceitos que nos permitem escrever programas em linguagem C que sejam capazes de realizar operações básicas, como: operações matemáticas, operações com a memória e com os dispositivos de entrada e saída. Além disso, lidar com fluxos diferentes dada uma determinada condição ou condições. Isso nos permite então cobrir parte dos problemas computacionais que surgem, mas ainda há situações que devem ser tratadas, dentre elas, a possibilidade do programa executar um número **n** de instruções. Exemplo, um dos exercícios realizados na aula 2 tem o objetivo de calcular o reajuste do salário de um funcionário conforme o seu cargo, na resolução do exercício é apresentado um programa que executa o cálculo para 1 funcionário e encerra, bem, imagine então uma situação hipotética em que a empresa tenha que fazer o cálculo do reajuste para 50 funcionários, neste caso ela terá que executar o programa 50 vezes manualmente, pois o programa apresentado como solução não é capaz de automatizar essa operação para todos os funcionários.

Então, para resolver esse problema computacional, é necessário lançar mão das estruturas de iteração que irão permitir que o nosso programa C tenha essa capacidade. Nesta aula serão apresentadas três estruturas de iteração, a estrutura **while**, a estrutura **do-while** e o **for**.

## 3.2 Cláusula for

A cláusula **for** é muito útil quando se deseja repetir uma ou várias instruções por um número **n** de vezes. Embora, o **for** possibilite variações, o formato de uso mais comum é utilizar uma variável que é incrementada e verificada a cada iteração, assim quando a variável atinge um determinado valor o laço irá encerrar (BACKES, 2013; SILVA; OLIVEIRA, 2014; MANZANO, 2015). Veja a seguir a sintaxe do **for**:

```
1 for (inicializacao; condicao de laço ou parada; incremento)
2   instrucao;
```

Na linha 1 da sintaxe apresentada foi incluída a palavra reservada **for** para indicar o início do laço, entre parênteses estão as definições do laço, é com base nestas definições que o comando irá definir quantas iterações serão realizadas. A primeira definição é a inicialização do variável que irá controlar o número de interações, a segunda definição é a condição que irá determinar se as iterações devem continuar ou não, a terceira definição é a forma como a variável irá incrementar. Na linha 2 foi adicionada a instrução que será executada **n** vezes. O **for** segue o mesmo padrão de outros comandos como o **if** que podem incluir blocos de instrução, ou seja, executar várias instruções, para isso, basta utilizar as chaves para indicar o início e término do bloco de instruções (KERNIGHAN; RITCHIE, 1988; BÄCKMAN, 2012; HASKINS, 2013). Veja a seguir:

```
1 for (inicializacao; condicao de laço ou parada; incremento) {
2   instrucao01;
3   instrucao02;
4   ...
5   instrucaoN;
6 }
```

Neste caso, é possível ver que, foi delimitado o início e término do bloco de instruções pelas chaves, e entre as chaves estão as **n** instruções que serão executadas no laço. A seguir um exemplo em forma de exercício para entender melhor este conceito.

### 3.2.1 Exercício de Exemplo

Faça um programa em C que leia 10 valores e ao final imprima a média aritmética dos valores lidos.

Fonte: Adaptado de [Lopes e Garcia \(2002, p. 152\)](#)

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 void main() {
4     float num, soma=0, media=0;
5     int i;
6     //inicio do laço for
7     for (i=1; i<=10; i++) {
8         //a partir deste ponto sao as instrucoes
9         //que devem ser executadas nas iteracoes
10        printf("Informe o numero:");
11        scanf("%f", &num);
12        soma += num;
13    }
14    //a media deve ser calculada apos a iteracao
15    media = soma / 10;
16    printf("A media e: %f", media);
17 }
```

Na linha 4 da resposta do exercício foram declaradas as variáveis **num** para armazenar o número a cada iteração, **soma** para acumular os valores que são inseridos pelo usuário e a variável **media** para armazenar o resultado do cálculo da média aritmética. Na linha 5 foi declarada a variável **i** do tipo inteiro, o objetivo ao declarar essa variável, foi controlar as iterações do **for**, e para isso, foi utilizado o tipo inteiro, pois é o tipo de número que será útil para contar as iterações, ou seja, pertencente ao conjunto dos inteiros,  $i = \{\dots, 0, 1, 2, 3, \dots\}$ . A cláusula **for** inicia na linha 7 em que são feitas as definições do laço. A primeira definição é a inicialização da variável **i** com o valor **1**, assim, foi definido que as iterações iniciarão com valor 1. A segunda definição, "**i<=10**", indica que as iterações serão executadas enquanto esta condição for **verdadeira**, ou seja, enquanto o conteúdo da variável é menor ou igual à 10. A terceira definição indica como será realizado o incremento da variável **i**, neste caso, "**i++**", quer dizer que a variável **i** vai ser incrementada de 1 em 1, então é possível concluir que este programa executará 10 iterações, uma vez que, **i** inicia com **1** e só termina quando for maior que **10**.

Entre as linhas 8 e 12 foram adicionadas as instruções que serão executadas pelo laço **for**, que no caso são: Linha 10 e 11, leitura do valor informado pelo usuário, como o programa deve ler 10 valores, essa instrução deve ser incluída no laço, linha 12, soma dos valores lidos. O objetivo de somar dentro do laço está associado à fórmula da média aritmética, pois, para calcular essa média é necessário somar primeiro todos os valores e depois dividir pela quantidade total dos valores. Desta forma, essa instrução deve ser incluída no laço, pois 10 valores deverão ser somados.

A linha 13 indica o término das instruções que serão executadas **n** vezes no laço **for**. Assim, a linha 15, traz a instrução responsável por dividir o total somado dos valores pela quantidade total de valores, que no caso é 10. A instrução na linha 16 é responsável pela impressão do resultado final. O exercício de exemplo, mostra claramente como é possível utilizar o **for** para executar várias iterações de uma ou várias rotinas. O exercício de exemplo pode nos levar a uma questão, e se não é conhecido previamente o número de iterações necessárias? Neste caso, pode-se utilizar uma variável para determinar o

número total de iterações. Para exemplificar, veja a seguir um exercício ajustado em relação ao exemplo anterior.

### 3.2.2 Exercício de Exemplo

Faça um programa em C que leia "**n**" valores. O programa deve inicialmente solicitar ao usuário que informe a quantidade desejada de valores a ser informada, depois ler os "**n**" valores e ao final imprimir a média aritmética dos valores lidos.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 void main() {
4     float num, soma=0, media=0;
5     int qtdeNum, i;
6     printf("Informe a quantidade de numeros:");
7     scanf("%d", &qtdeNum);
8     //inicio do laço for
9     for (i=1; i<=qtdeNum; i++) {
10        //a partir deste ponto são as instruções
11        //que devem ser executadas nas iterações
12        printf("Informe o numero:");
13        scanf("%f", &num);
14        soma += num;
15    }
16    //a media deve ser calculada após a iteração
17    media = soma / qtdeNum;
18    printf("A media é: %f", media);
19 }
```

Note na resposta do exercício que foram necessárias apenas poucas alterações para que o programa atenda à nova necessidade. Primeiro, na linha 5 foi adicionada a declaração da variável **qtdeNum** que será responsável por armazenar o total de números que devem ser lidos. Foram então adicionadas as linhas 6 e 7 para ler a quantidade total de números. A linha 9, com a cláusula **for** foi ajustada de forma que o valor 10 foi substituído pelo nome da variável **qtdeNum**, assim, ao invés da condição comparar com um valor fixo, a comparação será feita com o valor informado pelo usuário e armazenado na variável. Outra linha ajustada é a 17 em que o valor 10 foi novamente substituído pelo nome da variável que controla a quantidade, pois como não é conhecida a quantidade, a priori, este valor também deixa de ser fixo no cálculo da média.

Nos exemplos apresentados até o momento, o incremento da variável responsável pelo controle das iterações está sendo realizado de 1 em 1. Contudo, em alguns casos, pode ser útil fazer o incremento com outras proporções (KERNIGHAN; RITCHIE, 1988), exemplo, de 2 em 2, de 10 em 10, ou outras proporções conforme a necessidade do problema. A seguir um exemplo em forma de exercício.

### 3.2.3 Exercício de Exemplo

Faça um programa em C que imprima todos os valores entre 0 e 100 múltiplos de 10.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 void main() {
4     int i;
5     for (i=0; i<=100; i+=10) {
```

```
6     printf("Multiplo de 10: %d \n", i);
7 }
8 }
```

Primeiramente, é importante saber que existem outras soluções mais comuns para este problema, por exemplo, poderia verificar se um valor é múltiplo de 10 obtendo o resto da divisão dele. Contudo, a solução proposta teve a intenção de mostrar que é possível incrementar as iterações do laço **for** com valores diferentes do usual, que é 1. Note que na linha 5 o laço **for** recebeu como primeiro parâmetro a inicialização da variável "**i=0**", como segundo parâmetro o critério para a continuação das iterações, como sendo "**i<=100**" e como terceiro parâmetro, o incremento da variável **i**, como sendo "**i+=10**", ou seja, definiu-se que o incremento será de 10 em 10. Assim, é fácil imprimir apenas os valores múltiplos de 10 entre 0 e 100 como pode ser visto na solução do exercício.

Além de poder incrementar valores diferentes de 1, conforme visto, é possível também decrementar valores, permitindo assim, executar laços do final para o início, permitindo assim resolver facilmente alguns problemas computacionais que demandam este tipo de situação. A seguir um exercício para exemplificar.

### 3.2.4 Exercício de Exemplo

Faça um programa em C que imprima todos os valores entre 0 e 10 em ordem decrescente.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 void main() {
4     int i;
5     //inicio do laço for
6     for (i=10; i>=0; i--) {
7         printf("Numero: %d \n", i);
8     }
9 }
```

Como pode-se ver na resposta do exercício, a solução é muito simples, veja que na linha 6, a variável **i** inicia com o valor 10, isso porque é necessário decrementar a variável **i**, desta forma, foi preciso iniciar a variável de controle com um valor que permita realizar as subtrações necessárias. Se a variável **i** vai ser decrementada, então, a definição do controle de fluxo também deve mudar, note que o sinal de **<=** foi alterado para **>=**, pois agora, o sentido das iterações mudou para: do maior para o menor. Por fim, para que essa lógica funcione, **i++** foi trocado para **i-**, assim, está completa a solução que irá realizar as iterações no laço no sentido contrário.

É possível combinar laços com outras estruturas, como estrutura de decisão, desta forma, é possível, por exemplo, combinar o uso do **for** com o **if**. A seguir um exercício para exemplificar.

### 3.2.5 Exercício de Exemplo

Faça um programa em C que imprima todos os valores pares entre 1 e 20.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 void main() {
```

```
4 int i;  
5 for (i=1; i<=20; i++) {  
6     if ((i % 2) == 0)  
7         printf("Numero par: %d \n", i);  
8 }  
9 }
```

Na linha 5 da resposta proposta para o exercício, foram feitas as definições do laço **for**, iniciando com o valor 1 e terminando em 20 com incremento de 1 em 1. Como o exercício requer a impressão apenas dos números "**pares**", então, na linha 6 foi incluída a estrutura de decisão **if** dentro do bloco de instruções do **for**, ou seja, o **if** está aninhado ao **for**. Embora o **if** utilizado no exemplo está em sua forma mais simples, pode-se combinar o **for** com "**n**" estruturas em sua forma mais complexa.

### 3.3 Cláusula for com laços aninhados

A estrutura de iteração **for** permite também o uso de laços aninhados (encaixados) que são úteis quando são necessárias iterações dentro de outras. Não há limite de laços que pode-se aninhar a outros, contudo, quanto mais aninhamentos forem criados, menor será o desempenho do nosso algoritmo. O gráfico na figura 12 mostra a relação entre número de instruções que são executadas por número de laços aninhados para o caso em que o número de iterações do primeiro laço é 10.

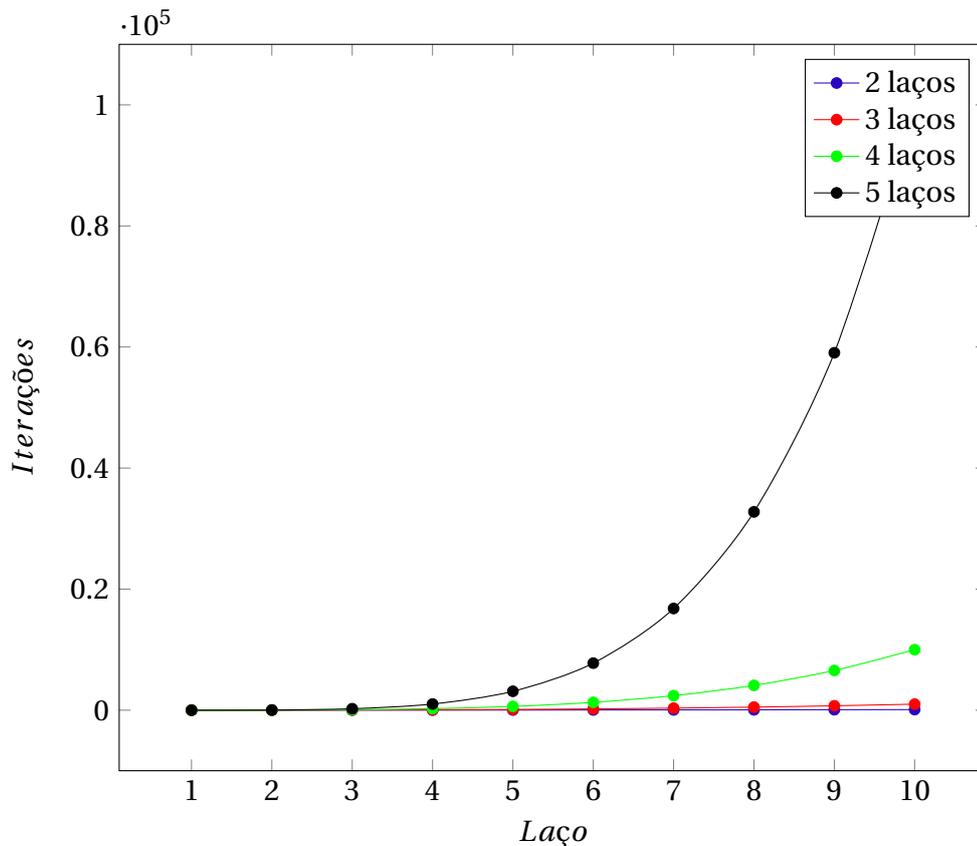


Figura 12 – N° de instruções executados x Laços aninhados

Veja na figura 12 que a partir de 4 laços aninhados o número de instruções cresce muito, isso porque o aumento é exponencial. Desta forma, ao escrever algoritmos em que haverá muitos laços aninhados, deve-se ter um cuidado especial neste caso, por exemplo, fazer testes suficientes para se certificar de que o tempo de execução será viável. É importante comentar também que, o gráfico mostra o número de instruções que serão executadas para cada caso, assumindo 10 iterações na primeira estrutura de iteração, contudo, o tempo de execução de cada instrução irá variar de acordo com o hardware utilizado. Outro ponto importante a ser destacado, é que, essa relação apresentada na figura 12 vale para qualquer estrutura de iteração, como: **for**, **while**, **do-while**. A seguir mais um exercício para exemplificar o uso de **for** aninhado.

### 3.3.1 Exercício de Exemplo

Faça um programa em C que imprima uma matriz de 4 linhas por 4 colunas, sendo que na primeira linha devem ser impressos os valores de 1 à 4 e partir da segunda linha, os valores impressos devem ser múltiplos da linha anterior.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 void main() {
4     int i, j;
5     //inicio do laço do primeiro for
6     for (i=1; i<=4; i++) {
7         //inicio do laço do segundo for
8         for (j=1; j<=4; j++) {
9             if (j < 4)
10                printf("%d \t", j*i);
11            else
12                printf("%d \n", j*i);
13        }
14    }
15 }
```

1	2	3	4
2	4	6	8
3	6	9	12
4	8	12	16

Figura 13 – Saída de exemplo de laço aninhado com **for**

Conforme pode ser visto na solução proposta para o exercício, embora aparentemente complexo, com o uso de laços aninhados é muito fácil resolver o exercício. O primeiro laço **for** na linha 6 é responsável pelas iterações nas linhas da matriz e o laço na linha 8 é responsável pela iteração nas colunas. Note que, como há dois laços aninhados, foi necessário utilizar duas variáveis de controle, pois caso fosse utilizada a mesma variável para controle dos dois laços ocorreriam problemas, pois um laço iria

alterar o valor da variável que estava com um determinado valor em outro laço, isso iria provocar efeitos inesperadas nos laços, assim, é sempre comum utilizar uma variável de controle para cada laço aninhado.

Na linha 9 utilizou-se um **if** para verificar se a última coluna da linha ainda não foi impressa, " $j < 4$ ", pois neste caso, o **printf** é combinado com o código "\t", pois assim, o valor é impresso e uma tabulação é adicionada, caso contrário o **printf** é combinado com o código "\n", para que uma quebra de linha seja adicionada e na próxima iteração do primeiro laço a impressão ocorra na próxima linha. Por fim, a multiplicação do **j** pelo **i** irá produzir exatamente os valores solicitados no exercício, pois a variável **j** sempre terá valores entre 0 e 4, uma vez que, a cada iteração do primeiro laço, o valor de **j** é reiniciado para 1, na primeira iteração de **i**,  $j * i$  irá resultar no próprio valor de **j**, pois **i** é 1, a partir da segunda iteração esse valor vai ser sempre múltiplo da linha anterior, uma vez que o valor é resultante de uma multiplicação. Na figura 13 pode ser visto o resultado da saída do programa.

### 3.4 Loop infinito na cláusula for

O uso de estruturas de iteração implica em repetir um determinado número de instruções, ou seja, um número finito de repetições, contudo, por erro de programação, é possível provocar um número infinito de repetições, em geral, o nome dado a este efeito é "loop infinito" ou "laço infinito". Essa situação é indesejável, pois uma vez provocada, o programa não conseguirá concluir a operação e o menor dos problemas será o estouro de memória do computador, ocasionando em travamento (KERNIGHAN; RITCHIE, 1988; SCHILDT, 1996; ANICHE, 2015). Desta forma, é necessário ter cuidado ao estabelecer a condição de saída do laço. Veja a seguir um exemplo de programa com loop infinito.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 void main() {
4     int i;
5     //inicio do laço
6     for (i=0; i<=0; i--) {
7         //instrucoes do bloco do laço
8         float j = i;
9         printf("Numero: %f \n", j);
10    }
11 }
```

Bem, está na linha 6 o código que provoca o laço infinito. Primeiro, verifique que a variável **i** é inicializada com zero, depois verifique o terceiro argumento, se trata de decremento, ou seja, a variável **i** é 0 na primeira iteração, -1 na segunda iteração, -2 na terceira iteração, etc. Por fim, veja o segundo argumento,  $i \leq 0$ , a condição diz que o laço irá se manter ativo enquanto a variável **i** é menor ou igual a zero, contudo, a variável **i** inicia com zero e decrementa a cada iteração, ou seja, **i** será sempre menor ou igual a zero, há então um loop infinito, pois a condição que mantém o laço ativo será sempre verdadeira, assim, o programa nunca sairá dessa estrutura de iteração, a menos, que o encerramento do programa seja forçado ou ocorra um estouro de memória. Desta forma, é preciso analisar com cuidado as definições que mantêm o laço ativo, pois em geral, os laços infinitos são provocados por erros neste aspecto.

## 3.5 Cláusula while

Diferente do **for** o **while** geralmente é empregado quando não se pode determinar com certeza quantas vezes um bloco de comandos será executado. A condição do **while** é definida de forma muito similar à definição da condição no **if**. A diferença é que no **if** o objetivo é desviar o caminho de execução para um fluxo de instruções ou outro, no **while** o objetivo será manter a execução de um bloco de instruções em execução, assim como no **for**. Veja a seguir a sintaxe **while**:

```
1 while (condicao de laço ou parada)
2     instrucao;
```

Na linha 1 da sintaxe apresentada foi incluída a palavra reservada **while** para indicar o início do laço, entre parênteses há a condição de laço ou parada, é com base nesta definição que o comando irá definir quantas iterações serão realizadas. Na linha 2 foi adicionada a instrução que será executada **n** vezes. O **while** segue o mesmo padrão de outros comandos como o **if** ou o **for** que podem incluir blocos de instrução, ou seja, executar várias instruções, para isso, basta utilizar as chaves para indicar o início e término do bloco de instruções (SCHILDT, 1996; ALBANO; ALBANO, 2010; GOOKIN, 2016). Veja a seguir:

```
1 while (condicao de laço ou parada) {
2     instrucao01;
3     instrucao02;
4     ...
5     instrucaoN;
6 }
```

Neste caso, é possível ver que, foi delimitado o início e término do bloco de instruções pelas chaves, e entre as chaves estão as "**n**" instruções que serão executadas no laço. O **while** avalia a condição definida em seu argumento, se o resultado da avaliação for **falso**, ou seja, retornar zero, então o laço termina e o programa continua na instrução seguinte ao **while**, se o resultado da avaliação da condição é **verdadeiro**, ou seja, diferente de zero, então as instruções no bloco de instruções do **while** são executadas, assim, enquanto a condição permanecer **verdadeira**, as instruções serão executadas. Veja um exemplo em forma de exercício para entender melhor.

### 3.5.1 Exercício de Exemplo

Faça um programa em C que realize a soma de todos os valores inteiros de 1 a **n**, sendo que **n** deve ser informado pelo usuário.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 void main() {
4     int i=1, n, soma=0;
5     printf("Informe o numero n:");
6     scanf("%d", &n);
7     while (i <= n) {
8         soma += i;
9         i++;
10    }
11    printf("Soma: %d \n", soma);
12 }
```

O exercício mostra um exemplo claro em que o número de iterações não pode ser determinado antes da execução do programa, pois o número de iterações é definido por **n** que deve ser informado pelo usuário, nesta situação é comum utilizar o **while** como estrutura de iteração. Veja que na linha 5 e 6 o usuário é solicitado à informar o "número n", pois conforme solicitado no exercício, o **n** é o dado de entrada do programa. Na linha 7 é iniciado o **while**, cuja condição que manterá o laço ativo está entre parênteses, **i <= n**, assim, se o usuário, por exemplo, informar o valor 5 para **n**, então o laço irá de 1 até 5, uma vez que **i** inicia com 1, conforme a linha 4. No caso do **while**, em geral, é preciso inicializar as variáveis responsáveis pelo controle do laço, por isso a variável **i** foi inicializada com 1 e a variável **n** foi inicializada pelo usuário.

O **while** verificará se a condição na linha 7 é **verdadeira**, ou seja, se **i** é menor que **n**, caso seja **verdadeira** então ele executará o bloco de instruções contido entre as chaves que são as linhas 8 e 9. A linha 8 é responsável pela soma de todos os valores entre 1 e **n**, e a linha 9 é responsável pelo incremento de **i**. Este ponto é importante destacar, note que no **for** o incremento da variável está presente entre os argumentos da cláusula **for**, isso porque no caso do **for**, sempre haverá o incremento de um contador, mas no caso do **while**, isso nem sempre é verdade, assim, é necessário fazer o incremento da variável no bloco de instruções do **while**. A linha 11 que é responsável pela impressão do resultado final foi colocada após o bloco de instruções do **while**, pois como o objetivo é somar todos os valores de 1 a **n**, então somente após a execução de todo o **while** é que a variável **soma** terá acumulado todos os valores.

Uma característica do **while** é a possibilidade de utilizar condições compostas assim como no **if**. O formato de uso segue o mesmo padrão do **if** requerendo o uso dos operadores lógicos de conjunção, disjunção e negação apresentados na tabela 9. As condições compostas são muito úteis em situações em que for necessário que o **while** avalie mais de uma condição para garantir a continuidade ou parada do laço. Não há limite para as condições a serem avaliadas, ou seja, pode-se incluir 2, 3, 4 ou **n** condições. Veja a seguir um exemplo de condição composta em **while**.

### 3.5.2 Exercício de Exemplo

Uma agência bancária de uma cidade do interior tem, no máximo, 10 mil clientes. Criar um programa em C que possa entrar com o número da conta, o nome e o saldo de cada cliente. Imprimir todas as contas, os respectivos saldos e uma das mensagens: positivo / negativo. A digitação acaba quando se digita -999 para número da conta ou quando chegar a 10 mil clientes. Ao final, deverá sair o total de clientes com saldo negativo, o total de clientes da agência e o saldo da agência.

Fonte: Adaptado de [Lopes e Garcia \(2002, p. 197\)](#)

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int conta, cTotNeg=0, cTot=0;
7     float saldo, soma=0;
8     char nome[30];
9     printf("Digite o numero da conta ou -999 para terminar:");
10    scanf("%d", &conta);
11    while (conta > 0 && cTot < 10000) {
12        cTot++;
13        printf("Nome:");
14        scanf("%s", &nome);
```

```
15     printf("Saldo:");
16     scanf("%f", &saldo);
17     soma += saldo;
18     if (saldo < 0) {
19         cTotNeg++;
20         printf("%d - %f - negativo \n", conta, saldo);
21     }
22     else {
23         printf("%d - %f - positivo \n", conta, saldo);
24     }
25     printf("Digite o numero da conta ou -999 para terminar:");
26     scanf("%d", &conta);
27 }
28 printf("\n Total de clientes com saldo negativo: %d", cTotNeg);
29 printf("\n Total de clientes da agencia: %d", cTot);
30 }
```

A resposta proposta para o exercício traz entre as linhas 6 e 8 a declaração das variáveis e a inicialização que se faz necessária. As linhas 9 e 10 são responsáveis pela leitura do valor da primeira conta a ser registrada, note que essa leitura é realizada antes de inicializar o **while**, pois como esta variável é utilizada na condição do **while** é necessário inicializar ela com algum valor, pois caso contrário, não há garantia que o programa executará o laço, pode até ocorrer de executar, pois em geral, variáveis não inicializadas trazem lixo de memória e esta informação pode favorecer a execução, mas não há garantia.

O ponto alto do exercício é na inicialização do **while** na linha 11, note que foram incluídas duas condições associadas pela conjunção **E (&&)**, como há uma conjunção, então as duas condições devem ser verdadeiras para que o bloco de instruções do **while** seja executado. A primeira condição, **conta > 0**, validará se o número informado para a conta é maior que zero, essa condição tem dois objetivos, o primeiro é garantir que valores válidos serão informados como número de conta, o segundo objetivo é oferecer à possibilidade do usuário encerrar a digitação antes do total de 10 mil contas, pois ao informar **-999**, esta condição será **falsa** forçando a parada do laço. A segunda condição, **cTot < 10000**, validará se o total de contas digitadas não atingiu o limite de 10 mil conforme solicitado no exercício.

As linhas seguintes, entre 12 e 24, são responsáveis pela leitura do nome do titular da conta, o saldo, a soma de todos os saldos e a impressão dos dados da conta com a informação de saldo positivo ou negativo, veja que entre as linhas 18 e 24 foi adicionado o uso do **if** no bloco do **while**, ou seja, assim como no **for**, é possível fazer uso de outras estruturas no bloco de instruções do **while** também. É importante notar as linhas 25 e 26, veja que são idênticas às 9 e 10, porque? Bem, como dito antes as linhas 9 e 10 estão antes da cláusula **while** em função da necessidade de inicializar a variável **conta**, contudo, a cada iteração do **while** é necessário novamente verificar se o usuário deseja continuar digitando novas contas ou se ele quer encerrar, por isso as linhas são repetidas no bloco de instruções do laço. Por fim, as linhas 28 e 29 são responsáveis pela impressão final dos resultados do programa. Aqui, chamo a atenção apenas ao fato de que a impressão deve ser realizada após o bloco de instruções do **while**, pois caso contrário o programa iria imprimir a cada iteração os valores acumulados nestas variáveis, o que não é desejado neste exercício.

## 3.6 Validação de dados com while

Na aula 2, estruturas de decisão, nos ocorreu uma situação em um determinado exercício que será discutida nesta aula. A situação envolvia o preenchimento de um valor dentro de uma faixa de valores, **1** para **Sim** e **0** para **Não**, para ser mais exato. Na ocasião a resolução proposta não incluía uma validação que forçasse o usuário a preencher apenas os valores dentro da faixa disponibilizada. Com os recursos aprendidos até agora, é possível resolver este problema incluindo essa validação com o **while**. A seguir um exemplo em formato de exercício.

### 3.6.1 Exercício de Exemplo

Faça um programa em linguagem C que leia 10 números positivos e imprima o quadrado de cada número. Para cada entrada de dados deverá haver um trecho de validação para que um número negativo não seja aceito pelo programa.

Fonte: Adaptado de [Lopes e Garcia \(2002, p. 192\)](#)

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main() {
5     float num;
6     int i;
7
8     for (i=1; i<=10; i++) {
9         printf("Informe um numero:");
10        scanf("%f", &num);
11        while (num <= 0) {
12            printf("\n ATENCAO! Informe um numero maior que zero:");
13            scanf("%f", &num);
14        }
15        printf("Quadrado: %f \n", num * num);
16    }
17 }
```

No exercício o ponto que nos interessa é a validação que garanta que os números digitados não sejam negativos. Nas linhas 5 e 6 foi incluída a declaração das variáveis, **num** para o armazenamento do número digitado pelo usuário e **i** para controle do laço **for**, pois como o número de iterações desejadas é conhecido, 10, essa será nossa opção. Na linha 8 foi adicionado o início do **for** e as definições necessárias para 10 iterações conforme solicitado no exercício. As linhas 9 e 10 são as instruções necessárias para a leitura do número e as linhas que seguem, entre 11 e 14, foi feita a validação do número lido, veja que o **while** será executado enquanto o número for menor ou igual à zero, **num <= 0**, ou seja, se o usuário digitar um número maior que zero, a análise da condição irá retornar **falso** e o bloco de instruções do **while** não irá executar, caso o usuário digite um número menor ou igual a zero, então as linhas 12 e 13 serão executadas e, portanto, a instrução na linha 12 irá alertar o usuário que ele deve digitar um número maior que zero e solicitar que digite novamente, e a linha 13 irá ler novamente um novo número digitado pelo usuário. Na sequência, se o usuário digitar novamente um número inferior ou igual a zero, a condição do **while** será novamente verdadeira e o bloco de instruções será novamente executado até que ele digite um número positivo ou encerre o programa. Por fim, após digitar um número positivo a instrução **while** será abortada e a linha 15 será executada imprimindo o quadrado do número lido.

## 3.7 Cláusula *while* com laços aninhados

Assim como na cláusula **for**, o **while** também permite o uso de **while** aninhado, ou seja, um laço **while** dentro de outro laço **while** ou mesmo outro laço **for** (para o **for**, vale a mesma regra, pode-se colocar um laço **while** no bloco de instruções do **for**), na verdade, não há limite para o número de laços que podem ser incluídos dentro de outro laço com aninhamento, contudo, vale a mesma regra apresentada na figura 12, desta forma, quanto maior for o número de laços **while** aninhados, menor será o desempenho do algoritmo, pois o aumento do número de instruções a serem executadas é exponencial. Assim, vale mais uma vez a regra de que deve-se ter cuidado ao usar este recurso. Analise a seguir, um exemplo de laço **while** aninhado por meio de um exercício.

### 3.7.1 Exercício de Exemplo

Na Usina de Angra dos Reis, os técnicos analisam a perda de massa de um material radioativo. Sabendo-se que este perde 25% de sua massa a cada 30 segundos, criar um programa em C que imprima o tempo necessário para que a massa deste material se torne menor que 0,10 grama. O programa deve calcular o tempo para várias massas.

Fonte: Adaptado de Lopes e Garcia (2002, p. 204)

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main() {
5     int conTempo = 0;
6     float massa, tempo;
7     char resp;
8
9     printf("Digite S se desejar novo calculo ou qualquer letra para terminar:");
10    scanf("%c", &resp);
11    while (resp == 'S' || resp == 's') {
12        printf("Digite a massa em gramas do material:");
13        scanf("%f", &massa);
14
15        while (massa >= 0.10) {
16            conTempo++;
17            massa *= 0.75;
18        }
19        tempo = (conTempo * 30) / 60;
20        printf("O tempo foi de: %f minutos. \n", tempo);
21        printf("\n Digite S se desejar novo calculo ou qualquer letra para
22            terminar:");
23        scanf(" %c", &resp);
24    }
```

Para resolver o exercício de exemplo foi necessário declarar 4 variáveis, na linha 5 a variável **conTempo**, responsável por contabilizar o tempo gasto para a perda de massa, foi necessário inicializar essa variável com zero, pois ela vai ser incrementada em um laço **while**. Na linha 6 foram declaradas as variáveis **massa** para receber e controlar a perda da massa e **tempo** para receber o resultado do cálculo do tempo. Na linha 7 foi declarada a variável **resp** para controlar a continuação da digitação da massa pelo usuário. As linhas 9 e 10 são responsáveis pela leitura da variável **resp**, o usuário deve informar **S** caso ele queira realizar o cálculo de perda de massa ou qualquer letra caso

ele não queira, isso porque na linha 11 a condição para o **while** executar a iteração é **resp == 'S'** ou **resp == 's'**, veja que neste caso foi necessário utilizar uma disjunção **OU** (**||**) para a condição composta na cláusula **while**. Essa condição composta é importante, pois o usuário pode vir a digitar a letra **S** em formato minúsculo o que resultaria em **falso** em uma comparação de igualdade com **S** em letra maiúsculo.

Ao validar como verdadeira a condição do **while** na linha 11 o bloco de instruções entre as linhas 12 e 22 será executado. Neste bloco de instruções, foi realizada a leitura da variável **massa** nas linhas 12 e 13, e na linha 15 o que espera-se exemplificar neste exercício, o **while** aninhado ao outro **while** da linha 11, este **while** interno realiza a contagem do tempo para a perda da massa até o alvo que é 0,10 gramas, por isso a condição do **while** é **massa >= 0.10**, ou seja, enquanto a massa é superior à 0,10 ela precisa incrementar o tempo gasto na linha 16 e sofrer perda de 25% na linha 17. Desta forma, há um laço aninhado com outro, o primeiro laço irá executar enquanto o usuário desejar realizar mais cálculos de perda de massa e o segundo laço irá executar, para cada cálculo de perda de massa, enquanto a perda não atinge o alvo de 0,10 gramas. Por fim, a linha 19 é responsável pela conversão do tempo em segundos, a linha 20 é responsável pela impressão do tempo calculado e convertido em segundos e as linhas 21 e 22 são responsáveis pela nova leitura da variável **resp** para que o usuário informe se deseja realizar mais um cálculo.

### 3.8 Loop infinito na cláusula while

Assim como na cláusula **for**, a estrutura **while** também está sujeita há erros de programação que levem ao *loop* ou laço infinito (KERNIGHAN; RITCHIE, 1988; SCHILDT, 1996; ANICHE, 2015). Como já mencionado o *loop* infinito é indesejado em nosso programa, desta forma, é preciso ter cuidado para não cometer erros que levem a este problema. A seguir um exemplo desta situação com a cláusula **while**.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main() {
5     int idade, resp=1;
6     while (resp == 1) {
7         printf("Digite a idade:");
8         scanf("%d", &idade);
9         printf("A idade e: %d \n", idade);
10    }
11    printf("\n Digite 1 para continuar ou outro numero para terminar:");
12    scanf("%d", &resp);
13 }
```

O exemplo é simples e o erro cometido é mais simples ainda, pois todas as linhas de código necessárias para que o programa funcione estão presentes, contudo a posição errada de duas linhas de código levam este programa há um *loop* infinito. Veja as linhas 11 e 12, essas duas linhas são responsáveis por alterar o valor da variável de controle do laço, **resp**, se essa variável é igual a **1**, quer dizer que o usuário deseja continuar digitando as idades, mas se o valor é diferente de **1**, então o programa deve abandonar o laço, mas como dito, no programa acima, o programa nunca irá abandonar o laço, apesar das instruções para mudar o conteúdo da variável estarem lá. Isso ocorre porque as duas linhas, 11 e 12 foram posicionadas no local errado, eles estão após o bloco de instruções do **while**, mas deveriam pertencer à este bloco, pois assim, a cada iteração o

usuário seria indagado se o mesmo deseja continuar digitando, como não é o caso, o laço repete, pois **resp** permanece igual a **1** e as linhas 11 e 12 nunca serão executadas. A seguir apresento a correção do código.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main() {
5     int idade, resp=1;
6     while (resp == 1) {
7         printf("Digite a idade:");
8         scanf("%d", &idade);
9         printf("A idade e: %d \n", idade);
10        printf("\n Digite 1 para continuar ou outro numero para terminar:");
11        scanf("%d", &resp);
12    }
13 }
```

Veja agora o que mudou, as linhas 11 e 12 no programa anterior agora pertencem ao bloco de instruções do **while** e passaram a ser as linhas 10 e 11, note que é um erro simples que pode ocorrer em função de uma chave que não foi notada, mas que poderá causar um bom estrago nos dados de um programa, dependendo do contexto. Basicamente, os erros que causam *loop* infinito no **while** estão relacionados à variável de controle do laço. Pode ser um esquecimento de atualizar o valor da variável ou mesmo uma comparação com um valor que nunca será atingido na condição do **while**, então basta ficar atento à estes possíveis problemas que os erros serão evitados.

## 3.9 Cláusula do-while

A cláusula **do-while** é diferente do **while** em um detalhe apenas. O bloco de instruções do **do-while** sempre executará ao menos uma vez, pois a condição no **do-while** é avaliada após a execução, assim, mesmo que a condição, após avaliada, seja **falsa**, o bloco já terá sido executado, desta forma, a condição **falsa** apenas impedirá uma nova execução do bloco de instruções. Assim, o **do-while** é recomendado quando não é conhecido previamente o número de iterações necessárias e será necessária a execução de pelo menos 1 iteração. Sabendo então que o **do-while** difere do **while** apenas neste aspecto, é possível aplicar ao **do-while** os mesmos recursos, como o uso de condições compostas e laços aninhados (SCHILDT, 1996; MIZRAHI, 2008; BACKES, 2013). Veja a seguir a sintaxe do **do-while**:

```
1 do {
2     instrucao;
3 } while (condicao de laco ou parada);
```

Como pode ser visto na sintaxe, o bloco de instruções, linha 2, é executado antes da condição, linha 3, ou seja, o **do-while** executa primeiro as instruções e depois verifica se deve ou não manter o laço ativo. Veja a seguir um exemplo em forma de exercício:

### 3.9.1 Exercício de Exemplo

Faça um programa em linguagem C que permita entrar com números e imprimir o quadrado de cada número digitado até entrar um número múltiplo de 6 que deverá ter

seu quadrado impresso também.

Fonte: Adaptado de [Lopes e Garcia \(2002, p. 189\)](#)

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int num;
7     do {
8         printf("\n Digite um numero ou multiplo de 6 para encerrar:");
9         scanf("%d", &num);
10        printf("Quadrado: %d \n", num * num);
11    } while ((num % 6) != 0);
12 }
```

O exercício pede que todos os valores lidos tenham o seu quadrado impresso, e define o ponto de parada do laço como sendo um número lido múltiplo de 6, contudo, como todos devem ser impressos, inclusive o múltiplo de 6, então há uma situação em que o **do-while** pode ser aplicado, pois o valor deve ser lido, calculado e impresso e só depois deve ser verificado se deve ou não sair do laço. Na resposta proposta, foi incluída a declaração da variável **num** na linha 6 e o **do-while** inicia na linha 7, as linhas 8, 9 e 10 são responsáveis pela leitura do número, cálculo do quadrado e impressão. Por fim, a linha 11 trata a condição de permanência do laço como sendo, **(num % 6) != 0**, ou seja, se o resto da divisão de 6 é diferente de zero, conclui-se que o número não é múltiplo de 6 e portanto as iterações devem continuar.

Assim, todas as operações são realizadas antes da verificação, se o número digitado é múltiplo de 6, então ele será calculado e impresso pelas instruções nas linhas anteriores e a análise da condição na linha 11 irá retornar falso fazendo com que o laço seja interrompido e a próxima linha, já fora do escopo do **do-while** seja executada. Dadas as características do **do-while**, é muito comum o seu uso para menus, veja a seguir um exemplo em formato de exercício.

### 3.9.2 Exercício de Exemplo

Faça um programa em linguagem C que funcione através do menu a seguir:

```

1- SOMA VARIOS NUMEROS

2- MULTIPLICA VARIOS NUMEROS

3- SAI DO PROGRAMA

OPCAO: |
```

Fonte: Adaptado de [Lopes e Garcia \(2002, p. 254\)](#)

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main() {
5     int op;
```

```
6 float num, soma, prod;
7
8 do {
9     printf("1-Soma varios numeros\n");
10    printf("2-Multiplica varios numeros\n");
11    printf("3-Encerrar o programa\n");
12    printf("Opcao:");
13    scanf("%d", &op);
14
15    if (op == 1) {
16        soma = 0;
17        printf("\n Digite numero ou -999 para finalizar:");
18        scanf("%f", &num);
19        while (num != -999) {
20            soma += num;
21            printf("\n Digite numero ou -999 para finalizar:");
22            scanf("%f", &num);
23        }
24        printf("\n Soma: %f \n", soma);
25    }
26    else if (op == 2) {
27        prod = 1;
28        printf("\n Digite numero ou -999 para finalizar:");
29        scanf("%f", &num);
30        while (num != -999) {
31            prod *= num;
32            printf("\n Digite numero ou -999 para finalizar:");
33            scanf("%f", &num);
34        }
35        printf("\n Produto: %f \n", prod);
36    }
37    else if (op == 3) {
38        printf("\n Programa encerrado!");
39    }
40    else {
41        printf("\n Opcao nao disponivel!");
42    }
43 } while (op != 3);
44 }
```

O exercício envolve programar um menu simples com 3 opções e, naturalmente, o comportamento irá variar de acordo com a opção, como o menu deve ser executado ao menos uma vez e não sabe-se, a priori, quantas vezes o menu será executado, então é uma situação em que é possível aplicar o **do-while**. Assim, as linhas 5 e 6 são responsáveis pela declaração das variáveis, a linha 8 possui a declaração inicial do **do-while**. Entre as linhas 9 e 13 foram adicionadas a impressão do menu e a leitura da opção do menu. Como a impressão do menu está no bloco de instruções do **do-while**, então é possível concluir que, primeiro, a impressão vai ocorrer ao menos uma vez e segundo, a cada vez que o usuário selecionar uma opção diferente de **3-Encerrar o programa** o menu será novamente impresso, pois a condição na linha 43 vai retornar verdadeiro e o laço permanecerá ativo. Note também que no bloco de instruções do **do-while** foram utilizados outros laços **while** para somar ou multiplicar vários números.

## 3.10 Exemplos adicionais

É comum utilizar as estruturas de iteração para realizar operações como: contar, somar, calcular a média, obter o mínimo e o máximo. A seguir foram disponibilizados dois exemplos adicionais em forma de exercício, um envolve as operações de contar, somar e calcular a média e o segundo exemplo envolve obter o mínimo e o máximo.

### 3.10.1 Exercício de Exemplo

Uma transportadora utiliza caminhões que suportam até 10 toneladas de peso, as caixas transportadas tem tamanho fixo e o caminhão comporta no máximo 200 volumes, assim, esta transportadora precisa controlar a quantidade e o peso dos volumes para acomodar nos caminhões. Faça um programa que leia **n** caixas e seu peso, ao final, o programa deve imprimir a quantidade de volumes, o peso total dos volumes e o peso médio dos volumes.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main() {
5     int resp, qtdVolumes=0;
6     float peso, pesoTotal=0, pesoMedio=0;
7     printf("Deseja cadastrar uma caixa? 1-SIM / 2-NAO \n");
8     scanf("%d", &resp);
9
10    while (resp == 1) {
11        qtdVolumes++;
12        printf("Informe o peso da caixa: \n");
13        scanf("%f", &peso);
14        pesoTotal += peso;
15        printf("Deseja cadastrar uma caixa? 1-SIM / 2-NAO \n");
16        scanf("%d", &resp);
17    }
18    pesoMedio = pesoTotal / qtdVolumes;
19
20    printf("Quantidade de volumes: %d \n", qtdVolumes);
21    printf("Peso total dos volumes: %f \n", pesoTotal);
22    printf("Peso medio dos volumes: %f \n", pesoMedio);
23 }
```

O exercício da transportadora pede que seja calculada a quantidade de volumes, calculado o peso total e a média dos pesos, ou seja, é uma situação de contar, somar e calcular a média. Outro detalhe importante é que neste exercício, não sabe-se a quantidade de volumes previamente, então é necessário utilizar o **while**. As linhas 5 e 6 são declarações das variáveis necessárias, note que foi criada uma variável para contar os volumes, **qtdVolumes**, uma variável para somar o peso, **pesoTotal** e uma variável para armazenar o peso médio, **pesoMedio**, essas variáveis estão relacionadas ao ponto chave da resolução do exercício. As linhas 7 e 8 são a leitura da variável **resp** que será utilizada no controle da condição do laço. Entre as linhas 10 e 17 foi adicionado o laço **while** que faz as três operações, lê o peso dos volumes, conta os volumes e soma o peso dos volumes.

Note que para contar a quantidade de volumes, foi necessário apenas inicializar a variável **qtdVolumes**, na linha 5 e incrementar a cada iteração na linha 11, pois o

número de iterações que o programa irá executar é exatamente igual ao número de volumes. Para somar os pesos, foi igualmente necessário inicializar a variável **pesoTotal** na linha 6 e acumular com a variável **peso** a cada iteração, na linha 14. Veja que para acumular com a variável **peso** é importante que essa instrução seja posicionada após a leitura do peso. O cálculo da média foi realizado fora do escopo do **while**, na linha 18, pois para calcular a média, primeiro é necessário somar todos os pesos e depois dividir pela quantidade de volumes. Para finalizar, foram incluídas as linhas 20, 21 e 22 que imprimem os resultados desejados.

### 3.10.2 Exercício de Exemplo

Num frigorífico existem 90 bois. Cada boi traz preso em seu pescoço um cartão contendo seu número de identificação e seu peso. Faça um programa que imprima a identificação e o peso do boi mais gordo e do boi mais magro (supondo que não haja empates).

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main() {
5     int i, idBoi, idBoiGordo, idBoiMagro;
6     float pesoBoi, boiGordo=0, boiMagro=0;
7
8     for (i=1; i<=90; i++) {
9         printf("Informe a identificacao do boi: \n");
10        scanf("%d", &idBoi);
11        printf("Informe o peso do boi: \n");
12        scanf("%f", &pesoBoi);
13
14        if (pesoBoi > boiGordo) {
15            idBoiGordo = idBoi;
16            boiGordo = pesoBoi;
17        }
18        if (pesoBoi < boiMagro || i == 1) {
19            idBoiMagro = idBoi;
20            boiMagro = pesoBoi;
21        }
22    }
23    printf("Identificacao do boi mais gordo: %d \n", idBoiGordo);
24    printf("Peso do boi mais gordo: %f \n", boiGordo);
25    printf("Identificacao do boi mais magro: %d \n", idBoiMagro);
26    printf("Peso do boi mais magro: %f \n", boiMagro);
27 }
```

No exemplo dos bois, o exercício pede que sejam lidos 90 bois, desta forma, é possível utilizar o **for**, pois sabe-se previamente que são 90, pede também que seja obtido e impresso o boi mais magro e o boi mais gordo, assim, trata-se de um caso de obter o mínimo e o máximo. Além disso, o exercício pede que seja impresso a identificação e o peso, seja ele do mais magro ou do mais gordo. As linhas 5 e 6 são as declarações das variáveis, note que, declarou-se a variável **idBoi** para ler a identificação do boi e **pesoBoi** para ler o peso do boi, essas duas variáveis é que irão armazenar temporariamente os dados de cada um dos 90 bois. Declarou-se também as variáveis: **idBoiGordo** e **boiGordo** para registrar os dados do boi mais gordo e **idBoiMagro** e **boiMagro** para registrar os dados do boi mais magro, essas são as variáveis chave da resolução do exercício.

A linha 8 do código proposto inicia o laço **for**, entre as linhas 9 e 12 é feita a leitura dos dados do boi da iteração atual. O ponto chave para resolver o exercício está entre as linhas 14 e 21, veja que para obter o boi mais gordo foi necessário inicializar a variável na linha 6 e foi realizada a validação do peso do boi lido com o boi registrado, **pesoBoi > boiGordo**, na linha 14, então analise este cenário! A variável **boiGordo** foi inicializada com zero, assumindo que não exista nenhum boi com peso menor ou igual a zero, então é possível concluir que qualquer peso de boi informado será superior a zero, então na primeira leitura o peso do primeiro boi lido irá substituir o valor da inicialização nas linhas 15 e 16. Nas iterações seguintes, o peso do novo boi lido só irá substituir o peso e identificação registrados, caso esse novo peso seja maior que o atual, assim, ao fim das iterações o peso do boi mais gordo será registrado na variável **boiGordo** e a sua respectiva identificação na variável **idBoiGordo**.

O mesmo procedimento é aplicado para o boi magro, foram realizadas apenas duas mudanças, uma é o sinal do operador relacional que passou de **maior (>)** para **menor (<)** e a outra é que foi adicionado na condição uma disjunção **OU ||** com a condição **i == 1**. Isso foi feito porque a variável **boiMagro** foi inicializada com zero, pois bem, não há nenhum boi com peso menor que zero, então a condição **i == 1** foi adicionada para garantir que se o laço estiver na primeira iteração ele vai executar as linhas 19 e 20 mesmo que o peso não seja menor, assim, garante-se que o valor inicial zero seja substituído pelo peso do primeiro boi lido, que até o momento será o de menor peso. Em caso de, após ele seja lido um boi com peso inferior, então o peso do novo boi vai ser armazenado na variável **boiMagro** garantindo que os dados do boi mais magro sejam registrados ao fim do laço **for**. Para finalizar o programa foram incluídas as linhas 23 à 26 para imprimir os resultados.

## 3.11 Resumo da Aula

Nesta aula foram apresentados os conceitos necessários para construir um programa em linguagem C com estruturas de iteração, ou seja, programas que possibilitem executar diversas tarefas repetidas vezes. Assim, o programa pode por exemplo, possibilitar o cálculo de um reajuste de salário de um número **n** de funcionários, onde **n** pode ser previamente conhecido ou não.

Neste sentido, foram apresentadas três cláusulas de estrutura de iteração em linguagem C, a cláusula **for**, **while** e **do-while**. A cláusula **for** é muito utilizada quando se conhece previamente o número de iterações necessárias no algoritmo, ou seja, quando é conhecido o **n**. Já as cláusulas **while** e **do-while**, em geral, são utilizadas quando não é conhecido o número de iterações, assim, essas cláusulas permitem estabelecer condições de parada que tornam possível a saída de um laço sem conhecer previamente o **n**. Contudo, há uma diferença entre as cláusulas **while** e **do-while**, no primeiro caso pode ocorrer de nenhuma iteração ser executada, no segundo caso, ao menos uma iteração será executada, isso ocorre porque enquanto no **while** a validação da condição é realizada antes da execução das instruções do bloco, no **do-while** a validação é realizada após a execução.

Não há como dizer que uma cláusula é melhor que a outra, pois de uma forma geral e com poucas adaptações, em geral se consegue aplicar o uso das 3 cláusulas em qualquer problema computacional que envolva a repetição de instruções. Contudo, de acordo com o contexto do problema, é possível identificar qual das três cláusulas se aplica melhor. Desta forma, ao escolher corretamente a cláusula mais apropriada, é provável que o programador irá escrever o código mais eficiente.

## 3.12 Exercícios da Aula

Os exercícios desta lista foram Adaptados de [Lopes e Garcia \(2002, p. 136-226\)](#).

1. Faça um programa em C que imprima todos os números de 1 até 100.
2. Faça um programa que imprima todos os números pares de 100 até 1.
3. Faça um programa que imprima os múltiplos de 5, no intervalo de 1 até 500.
4. Faça um programa em C que permita entrar com o nome, a idade e o sexo de 20 pessoas. O programa deve imprimir o nome da pessoa se ela for do sexo masculino e tiver mais de 21 anos.
5. Sabendo-se que a unidade lógica e aritmética calcula o produto através de somas sucessivas, crie um programa que calcule o produto de dois números inteiros lidos. Suponha que os números lidos sejam positivos e que o multiplicando seja menor do que o multiplicador.
6. Crie um programa em C que imprima os 20 primeiros termos da série de Fibonacci.  
**Observação:** os dois primeiros termos desta série são 1 e 1 e os demais são gerados a partir da soma dos anteriores. Exemplo:
  - $1 + 1 = 2$ , terceiro termo;
  - $1 + 2 = 3$ , quarto termo, etc.
7. Crie um programa em linguagem C que permita entrar com o nome, a nota da prova 1 e da prova 2 de 15 alunos. Ao final, imprimir uma listagem, contendo: nome, nota da prova 1, nota da prova 2, e média das notas de cada aluno. Ao final, imprimir a média geral da turma.
8. Faça um programa que permita entrar com o nome e o salário bruto de 10 pessoas. Após ler os dados, imprimir o nome e o valor da alíquota do imposto de renda calculado conforme a tabela a seguir:

Salário	IRRF
Salário menor que R\$1300,00	Isento
Salário maior ou igual a R\$1300,00 e menor que R\$2300,00	10% do salário bruto
Salário maior ou igual a R\$2300,00	15% do salário bruto

9. No dia da estréia do filme "Procurando Dory", uma grande emissora de TV realizou uma pesquisa logo após o encerramento do filme. Cada espectador respondeu a um questionário no qual constava sua idade e a sua opinião em relação ao filme: excelente - 3; bom - 2; regular - 1. Criar um programa que receba a idade e a opinião de 20 espectadores, calcule e imprima:
  - A média das idades das pessoas que responderam excelente;
  - A quantidade de pessoas que responderam regular;
  - A percentagem de pessoas que responderam bom entre todos os expectadores analisados.
10. Em um campeonato Europeu de Volleyball, se inscreveram 30 países. Sabendo-se que na lista oficial de cada país consta, além de outros dados, peso e idade de 12 jogadores, crie um programa que apresente as seguintes informações:

- O peso médio e a idade média de cada um dos times;
  - O atleta mais pesado de cada time;
  - O atleta mais jovem de cada time;
  - O peso médio e a idade média de todos os participantes.
11. Construa um programa em C que leia vários números e informe quantos números entre 100 e 200 foram digitados. Quando o valor 0 (zero) for lido, o algoritmo deverá cessar sua execução.
  12. Dado um país A, com 5 milhões de habitantes e uma taxa de natalidade de 3% ao ano, e um país B com 7 milhões de habitantes e uma taxa de natalidade de 2% ao ano, fazer um programa que calcule e imprima o tempo necessário para que a população do país A ultrapasse a população do país B.
  13. Uma empresa de fornecimento de energia elétrica faz a leitura mensal dos medidores de consumo. Para cada consumidor, são digitados os seguintes dados:
    - número do consumidor
    - quantidade de kWh consumidos durante o mês
    - tipo (código) do consumidor
      - 1-residencial, preço em reais por kWh = 0,3
      - 2-comercial, preço em reais por kWh = 0,5
      - 3-industrial, preço em reais por kWh = 0,7

Os dados devem ser lidos até que seja encontrado o consumidor com número 0 (zero). O programa deve calcular e imprimir:

- O custo total para cada consumidor
  - O total de consumo para os três tipos de consumidor
  - A média de consumo dos tipos 1 e 2
14. Faça um programa que leia vários números inteiros e apresente o fatorial de cada número. O algoritmo encerra quando se digita um número menor do que 1.
  15. Faça um programa em C que permita entrar com a idade de várias pessoas e imprima:
    - total de pessoas com menos de 21 anos
    - total de pessoas com mais de 50 anos
  16. Sabendo-se que a unidade lógica e aritmética calcula a divisão por meio de subtrações sucessivas, criar um algoritmo que calcule e imprima o resto da divisão de números inteiros lidos. Para isso, basta subtrair o divisor ao dividendo, sucessivamente, até que o resultado seja menor do que o divisor. O número de subtrações realizadas corresponde ao quociente inteiro e o valor restante da subtração corresponde ao resto. Suponha que os números lidos sejam positivos e que o dividendo seja maior do que o divisor.
  17. Crie um programa em C que possa ler um conjunto de pedidos de compra e calcule o valor total da compra. Cada pedido é composto pelos seguintes campos:
    - número de pedido

- data do pedido (dia, mês, ano)
- preço unitário
- quantidade

O programa deverá processar novos pedidos até que o usuário digite 0 (zero) como número do pedido.

18. Uma pousada estipulou o preço para a diária em R\$30,00 e mais uma taxa de serviços diários de:
- R\$15,00, se o número de dias for menor que 10;
  - R\$8,00, se o número de dias for maior ou igual a 10;

Faça um programa que imprima o nome, a conta e o número da conta de cada cliente e ao final o total faturado pela pousada.

O programa deverá ler novos clientes até que o usuário digite 0 (zero) como número da conta.

19. Em uma Universidade, os alunos das turmas de informática fizeram uma prova de algoritmos. Cada turma possui um número de alunos. Criar um programa que imprima:
- quantidade de alunos aprovados;
  - média de cada turma;
  - percentual de reprovados.

**Obs.:** Considere aprovado com nota  $\geq 7.0$

20. Uma pesquisa de opinião realizada no Rio de Janeiro, teve as seguintes perguntas:
- Qual o seu time de coração?
    - 1-Fluminense;
    - 2-Botafogo;
    - 3-Vasco;
    - 4-Flamengo;
    - 5-Outros
  - Onde você mora?
    - 1-RJ;
    - 2-Niterói;
    - 3-Outros
  - Qual o seu salário?

Faça um programa que imprima:

- o número de torcedores por clube;
- a média salarial dos torcedores do Botafogo;
- o número de pessoas moradoras do Rio de Janeiro, torcedores de outros clubes;
- o número de pessoas de Niterói torcedoras do Fluminense

**Obs.:** O programa encerra quando se digita 0 para o time.

21. Em uma universidade cada aluno possui os seguintes dados:

- Renda pessoal;
- Renda familiar;
- Total gasto com alimentação;
- Total gasto com outras despesas;

Faça um programa que imprima a porcentagem dos alunos que gasta acima de R\$200,00 com outras despesas. O número de alunos com renda pessoal maior que a renda familiar e a porcentagem gasta com alimentação e outras despesas em relação às rendas pessoal e familiar.

**Obs.:** O programa encerra quando se digita 0 para a renda pessoal.

22. Crie um programa que ajude o DETRAN a saber o total de recursos que foram arrecadados com a aplicação de multas de trânsito.

O algoritmo deve ler as seguintes informações para cada motorista:

- número da carteira de motorista (de 1 a 4327);
- número de multas;
- valor de cada uma das multas.

Deve ser impresso o valor da dívida para cada motorista e ao final da leitura o total de recursos arrecadados (somatório de todas as multas). O programa deverá imprimir também o número da carteira do motorista que obteve o maior número de multas.

**Obs.:** O programa encerra ao ler a carteira de motorista de valor 0.

23. Crie um programa que leia um conjunto de informações (nome, sexo, idade, peso e altura) dos atletas que participaram de uma olimpíada, e informar:

- a atleta do sexo feminino mais alta;
- o atleta do sexo masculino mais pesado;
- a média de idade dos atletas.

**Obs.:** Deverão se lidos dados dos atletas até que seja digitado o nome @ para um atleta.

**Para resolver este exercício, consulte a aula 7** que aborda o tratamento de strings, como comparação e atribuição de textos.

24. Faça um programa que calcule quantos litros de gasolina são usados em uma viagem, sabendo que um carro faz 10 km/litro. O usuário fornecerá a velocidade do carro e o período de tempo que viaja nesta velocidade para cada trecho do percurso. Então, usando as fórmulas **distância = tempo x velocidade** e **litros consumidos = distância / 10**, o programa computará, para todos os valores não-negativos de velocidade, os litros de combustível consumidos. O programa deverá imprimir a distância e o número de litros de combustível gastos naquele trecho. Deverá imprimir também o total de litros gastos na viagem. O programa encerra quando o usuário informar um valor negativo de velocidade.

25. Faça um programa que calcule o imposto de renda de um grupo de contribuintes, considerando que:
- os dados de cada contribuinte (CIC, número de dependentes e renda bruta anual) serão fornecidos pelo usuário via teclado;
  - para cada contribuinte será feito um abatimento de R\$600 por dependente;
  - a renda líquida é obtida diminuindo-se o abatimento com os dependentes da renda bruta anual;
  - para saber quanto o contribuinte deve pagar de imposto, utiliza-se a tabela a seguir:

Renda Líquida	Imposto
até R\$1000	Isento
de R\$1001 a R\$5000	15%
acima de R\$5000	25%

- o valor de CIC igual a zero indica final de dados;
  - o programa deverá imprimir, para cada contribuinte, o número do CIC e o imposto a ser pago;
  - ao final o programa deverá imprimir o total do imposto arrecadado pela Receita Federal e o número de contribuintes isentos;
  - leve em consideração o fato de o primeiro CIC informado poder ser zero.
26. Foi feita uma pesquisa de audiência de canal de TV em várias casas de uma certa cidade, em um determinado dia. Para cada casa visitada foram fornecidos o número do canal (4, 5, 7, 12) e o número de pessoas que estavam assistindo a ele naquela casa. Se a televisão estivesse desligada, nada seria anotado, ou seja, esta casa não entraria na pesquisa. Criar um programa que:
- Leia um número indeterminado de dados, isto é, o número do canal e o número de pessoas que estavam assistindo;
  - Calcule e imprima a porcentagem de audiência em cada canal.

**Obs.:** Para encerrar a entrada de dados, digite o número do canal zero.

27. Crie um programa que calcule e imprima o CR do período para os alunos de computação. Para cada aluno, o algoritmo deverá ler:
- número da matrícula;
  - quantidade de disciplinas cursadas;
  - notas em cada disciplina;
- Além do CR de cada aluno, o programa deve imprimir o melhor CR dos alunos que cursaram 5 ou mais disciplinas.
- fim da entrada de dados é marcada por uma matrícula inválida (matrículas válidas de 1 a 5000);
  - CR do aluno é igual à média aritmética de suas notas.
28. Construa um programa que receba a idade, a altura e o peso de várias pessoas, Calcule e imprima:

- a quantidade de pessoas com idade superior a 50 anos;
  - a média das alturas das pessoas com idade entre 10 e 20 anos;
  - a porcentagem de pessoas com peso inferior a 40 quilos entre todas as pessoas analisadas.
29. Construa um programa que receba o valor e o código de várias mercadorias vendidas em um determinado dia. Os códigos obedecem a lista a seguir:

L-limpeza

A-Alimentação

H-Higiene

**Calcule e imprima:**

- o total vendido naquele dia, com todos os códigos juntos;
- o total vendido naquele dia em cada um dos códigos.

**Obs.:** Para encerrar a entrada de dados, digite o valor da mercadoria zero.

30. Faça um programa que receba a idade e o estado civil (C-casado, S-solteiro, V-viúvo e D-desquitado ou separado) de várias pessoas. Calcule e imprima:

- a quantidade de pessoas casadas;
- a quantidade de pessoas solteiras;
- a média das idades das pessoas viúvas;
- a porcentagem de pessoas desquitadas ou separadas dentre todas as pessoas analisadas.

**Obs.:** Para encerrar a entrada de dados, digite um número menor que zero para a idade.



# AULA 4

## Vetores

### Metas da Aula

1. Entender e praticar os conceitos do uso de vetores na linguagem C.
2. Aplicar variadas situações relacionadas ao uso de vetores em programação.
3. Escrever programas que farão uso de vetores.

### Ao término desta aula, você será capaz de:

1. Declarar um vetor.
2. Obter os valores de variadas posições de um vetor.
3. Atribuir valores em variadas posições no vetor.
4. Utilizar estrutura de iteração com vetores.

## 4.1 Vetores

Até o momento, foram utilizadas variáveis em que é possível armazenar 1 (um) dado considerando o seu tipo, como pode ser visto na tabela 10, assim, se for declarada uma variável do tipo **int**, por exemplo, o programa irá reservar **16 bits** na memória do computador para essa variável e com este espaço, pode-se armazenar um dado apenas, na memória, isso ocorre porque a variável está associada à apenas aquela área da memória que foi reservada para ela. Assim, se for necessário armazenar, por exemplo, a idade de 2 pessoas, duas variáveis devem ser declaradas, se for necessário armazenar a idade de 3 pessoas, então 3 variáveis devem ser declaradas, mas e se for necessário armazenar a idade de 10 ou 100 pessoas? Não seria nada prático declarar 10 variáveis, muito menos, declarar 100 variáveis, até mesmo porque o maior esforço seria no controle dessas diversas variáveis. Neste caso é muito útil o uso de vetores.

Tabela 10 – Tamanho dos tipos básicos na linguagem C

<i>Tipo</i>	<i>Tamanho em bits</i>	<i>Faixa</i>
<b>int</b>	16	-32.767 a 32.767
<b>long int</b>	32	-2.147.483.647 a 2.147.483.647
<b>short int</b>	16	o mesmo que <b>int</b> .
<b>float</b>	32	Seis dígitos de precisão
<b>double</b>	64	Dez dígitos de precisão
<b>char</b>	8	-127 a 127

Fonte: Adaptado de Schildt (1996, p. 17)

Os **vetores**, também denominados de **arrays**, são um conjunto de variáveis do mesmo tipo que facilitam a manipulação dos dados, pois permitem o acesso por meio de índices (ALBANO; ALBANO, 2010; BACKES, 2013; ANICHE, 2015). Ao declarar um vetor, é preciso informar o seu tamanho e o seu tipo, por exemplo, um vetor do tipo **int** com tamanho 10, indica que este vetor terá uma capacidade de armazenar até 10 valores do tipo inteiro, pois ele faz referência para 10 posições de área reservada na memória com 16 bits, conforme pode ser visto na tabela 10. Os vetores são estruturas de um tipo apenas, ou seja, se for declarado um vetor do tipo **int**, todas as suas posições serão do tipo **int**, se o vetor é do tipo **float**, então todas as suas posições são do tipo **float**, assim, os vetores são estruturas homogêneas. Veja a seguir a sintaxe para a declaração de um vetor.

```
1 tipoVetor nomeVetor[tamanho];
```

Os tipos disponíveis para a declaração de um vetor seguem o mesmo padrão apresentado na tabela 1, além disso, para o nome do vetor devem ser seguidas as mesmas regras para a declaração do nome da variável, como explicado na aula 1, alguns exemplos de erros que não podem ser cometidos ao declarar o nome da variável e vetor são apresentados na tabela 2. Veja a seguir alguns exemplos de declaração de vetores.

```
1 int idade[10];
2 float notas[5];
3 double preco[20];
```

Na linha 1 do exemplo, foi declarado um vetor do tipo **int** com o nome **idade** e tamanho 10. Assim, este vetor poderá armazenar até 10 valores do tipo inteiro. Na linha 2 foi declarado o vetor **notas** do tipo **float** com tamanho igual a 5, desta forma,

sua capacidade permite armazenar 5 valores do conjunto dos reais, e na linha 3 foi declarado o vetor **preco** do tipo **double** com tamanho igual a 20 que permite armazenar 20 valores do conjunto dos reais.

## 4.2 Atribuição e obtenção de valores em vetor

A atribuição em um vetor é muito simples e fácil de aprender, mas antes de avançar, analise a figura 14. Esta figura traz uma representação gráfica, para fins didáticos, da declaração de um vetor de 10 posições. Na prática, ao declarar um vetor, ocorre a reserva de uma área da memória que seja capaz de acomodar as posições do vetor em sequência, assim, a primeira posição terá um endereço de memória e as demais posições terão os endereços subsequentes. Mas, para o programador a área do vetor deve ser acessada como na figura 14, note que na figura cada posição do vetor, representada por um quadrado, possui um número na parte inferior, começando de zero e terminando com 9. Esses números são os índices que devem ser utilizados para acessar o vetor.

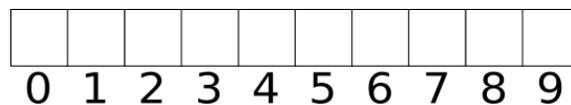


Figura 14 – Representação de um vetor de 10 posições

Assim, ao declarar um vetor no C, a posição inicial será **0 (zero)** e a posição final será **n-1**. Exemplo, para um vetor de 10 posições, a posição inicial é **0** e a final é **n-1**, ou seja, **10-1**, que é **9**. Da mesma forma, para um vetor de tamanho igual a **100**, a posição inicial é **0** e a posição final é **99**. Ao entender este conceito, a atribuição e obtenção de valores em vetores se torna fácil. Para atribuir um valor em um vetor, primeiro é preciso estar claro que, como o vetor possui **n** posições, então a atribuição é realizada a cada posição individualmente, ou seja, em um vetor com 10 posições serão necessárias 10 instruções para atribuir valor a todas as posições. A atribuição é realizada da mesma forma que é feita em uma variável, a diferença está no uso do índice. Veja a seguir a sintaxe:

```
1 nomeVetor[posicao] = valor;
```

Em **nomeVetor**, substitua pelo nome do vetor definido, em **posicao**, substitua pelo índice da posição em que se deseja armazenar o valor, em **valor** substitua pelo valor desejado, que assim como em atribuição de variável, pode ser uma expressão matemática, o valor armazenado em outra variável ou um valor fixo no código-fonte. Outra forma de fazer atribuição de valor a uma posição de um vetor é com a função **scanf**. Neste caso, a forma de uso é idêntica à da variável, mas, novamente deve ser adicionado o índice da posição do vetor. Veja a seguir um trecho de código-fonte com exemplos de atribuição em vetor.

```
1 //declarando um vetor com tamanho 10
2 int vetIdades[10];
3 //atribuindo um valor na primeira posicao
4 vetIdades[0] = 10;
5 //atribuindo um valor na segunda posicao
6 vetIdades[1] = 20;
7 //atribuindo um valor na terceira posicao com scanf
8 printf("Informe uma idade: \n");
```

```

9 scanf("%d", &vetIdades[2]);
10 //atribuindo um valor na quarta posicao
11 vetIdades[3] = vetIdades[2] + vetIdades[1];

```

No código de exemplo o vetor **vetIdades** foi declarado na linha 2 com o tipo **int** e tamanho 10, na linha 4 foi realizada uma atribuição do valor **10** na primeira posição do vetor, cujo índice é **0 (zero)**. Na linha 6 foi realizada a atribuição do valor **20** na segunda posição do vetor, visto que a primeira posição tem índice 0, então é natural que a segunda posição tenha índice 1. Na linha 9 do código foi realizada uma atribuição com o uso da função **scanf**, note que segue o mesmo padrão de uso com variável, o que muda é apenas a indicação do índice da posição em que o valor deve ser armazenado. A linha 11 é um exemplo de atribuição com expressão matemática, em que, foram utilizados os valores já presentes no próprio vetor, índices 2 e 1 respectivamente, para a soma.

Assim como na atribuição, a obtenção de um valor de um vetor segue os mesmos moldes que em uma variável, exceto, o fato de que deve-se indicar o índice da posição. No código de exemplo já apresentado, a linha 11 é um caso de obtenção de valor, pois na expressão, os valores armazenados nos índices 2 e 1 foram obtidos para a realização do cálculo. Veja na figura 15 uma representação didática de como seria o resultado do vetor **vetIdades** após a execução do código de exemplo. Note que, nos índices 0, 1 e 2 foram armazenados valores, os demais índices permanecem vazios, assim, para obter, por exemplo, o valor **20** deve ser acessado o índice **1**, para obter o valor **30** deve ser acessado o índice **2**. Veja a seguir mais um trecho de código-fonte com exemplos de obtenção do valor em posições do vetor.

vetIdades									
10	20	30							
0	1	2	3	4	5	6	7	8	9

Figura 15 – Representação de um vetor com alguns valores armazenados

```

1 //declarando um vetor com tamanho 3 e uma variavel soma
2 int vetIdades[3], soma;
3 //atribuindo valores com scanf
4 printf("Informe a primeira idade: \n");
5 scanf("%d", &vetIdades[0]);
6 printf("Informe a segunda idade: \n");
7 scanf("%d", &vetIdades[1]);
8 printf("Informe a terceira idade: \n");
9 scanf("%d", &vetIdades[2]);
10
11 soma = vetIdades[0] + vetIdades[1] + vetIdades[2];
12
13 printf("Idades: %d \t %d \t %d \n", vetIdades[0], vetIdades[1], vetIdades[2]);
14 printf("A soma das idades e: %d \n", soma);

```

Na linha 2 do código de exemplo foi declarado o vetor **vetIdades** com tamanho 3 e a variável **soma**, note que uma variável pode ser declarada na mesma linha de um vetor quando os tipos são iguais. Entre as linhas 4 e 9 foram atribuídos valores ao vetor com o uso da função **scanf**. Na linha 11 foram obtidos os valores das 3 posições do vetor e somados, e o resultado foi atribuído à variável **soma**. A linha 13 é um exemplo de

obtenção do valor de uma posição do vetor para impressão na tela com o **printf**, veja que o formato de uso é igual ao de uma variável, com a diferença apenas na indicação do índice da posição do vetor.

A indicação correta dos índices do vetor ao acessar ou atribuir valores é de suma importância, pois cada posição possui uma referência para uma área da memória, desta forma, ao tentar acessar um índice que não pertence àquele vetor o resultado pode gerar erros dos mais variados, pois provavelmente, o programa tentará acessar um trecho de memória que possivelmente é válido, mas que pode pertencer a uma outra variável, ou mesmo a um outro programa (BACKES, 2013). Assim, certifique-se de sempre delimitar corretamente a faixa de índices ao acessar o vetor.

## 4.3 Atribuição e acesso a valores com estrutura de iteração

Como pode ver, para atribuir ou acessar valores em vetores é preciso uma instrução para cada posição do vetor. Assim, em vetores com poucas posições pode até ser prático fazer o acesso, contudo imagine fazer a atribuição em um vetor com 100 posições? Terá que escrever 100 instruções no código-fonte? Certamente, isso não é nada prático, nestes casos é possível fazer uso das estruturas de iteração e a que se adapta melhor com os vetores é o **for**. Isso ocorre porque o **for** já traz um índice que pode facilmente ser utilizado para indicar as posições do vetor. A seguir, um exercício para exemplificar.

### 4.3.1 Exercício de Exemplo

Faça um programa em C que armazene 15 números inteiros em um vetor NUM e imprima uma listagem dos números lidos.

Fonte: Adaptado de Lopes e Garcia (2002, p. 278)

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main() {
5     int i, NUM[15];
6
7     //lendo os valores
8     for (i=0; i<15; i++) {
9         printf("Informe um numero: \n");
10        scanf("%d", &NUM[i]);
11    }
12    //imprimindo os valores
13    for (i=0; i<15; i++) {
14        printf("Numero: %d \n", NUM[i]);
15    }
16 }
```

O código-fonte proposto como resposta do exercício mostra como é possível utilizar uma estrutura de iteração, no caso o **for**, para facilitar o trabalho ao utilizar vetores. Veja que na linha 5 foi declarada a variável **i** para uso como índice do laço e do vetor, foi declarada também a variável **NUM** com o tipo **int** e tamanho 15 para atender ao solicitado no enunciado do exercício. Assim, o vetor **NUM** irá armazenar os 15 valores inteiros lidos. Para ler facilmente os valores, foi utilizado um **for** na linha 8, note que, para facilitar o uso do **i** como índice do vetor, o mesmo foi inicializado com **0 (zero)**,

e como a última posição do vetor **NUM** tem índice 14, a condição de parada do laço é **i<15**, ou seja, as iterações irão ocorrer até **i** atingir o valor 14. Na linha 10, veja que o **i** foi utilizado como índice do vetor, pois os valores incrementados em **i** serão compatíveis com os índices de **NUM**, permitindo assim o seu uso como índice tanto no laço quanto no vetor. Entre as linhas 13 e 15, há um segundo **for**, pois neste caso, o objetivo é após ler todos os valores, nas linhas 8 à 11, imprimir todos eles, o que é realizado na linha 14 pelo **printf**. Veja um outro exemplo de uso de estrutura de iteração em um exercício.

### 4.3.2 Exercício de Exemplo

Faça um programa em C que leia 5 valores inteiros e armazene em um vetor. Após a leitura o programa deve encontrar e imprimir o maior valor armazenado no vetor.

Fonte: Adaptado de Backes (2013, p. 127)

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main() {
5     int i, lista[5];
6
7     //lendo os valores
8     for (i=0; i<5; i++) {
9         printf("Informe um numero: \n");
10        scanf("%d", &lista[i]);
11    }
12    //inicializando o maior valor
13    int maior = lista[0];
14    //encontrando o maior valor
15    for (i=1; i<5; i++) {
16        if (maior < lista[i])
17            maior = lista[i];
18    }
19    printf("Maior = %d \n", maior);
20 }
```

Este exercício é um exemplo bem interessante para avaliar, pois além exemplificar a atribuição e acesso à valores em um vetor, mostra também como avaliar os valores, no caso, com o objetivo de encontrar o maior valor. A resolução do exercício proposta traz na linha 5 a declaração da variável **i** e o vetor **lista** com tamanho 5. Entre as linhas 8 e 11 é feita a leitura dos valores e armazenado no vetor **lista**. Foi necessário declarar uma variável **maior** para guardar o maior valor, pois assim, é possível acessar todos os valores do vetor sem perder dados. Cabe notar também na linha 13 que a variável **maior** também foi inicializada com o valor da primeira posição do vetor **lista**, porque? O objetivo é armazenar o primeiro valor, e acessar os valores seguintes, por meio do **for** entre as linhas 15 e 18, verificando sempre, se o próximo valor acessado é maior que o valor atual, na linha 16, caso seja, o valor é então substituído na linha 17, pois este passa a ser o maior valor, caso não seja, o programa apenas avança para o próximo valor até terminar as posições do vetor, ao final da iteração, o maior valor do vetor estará armazenado na variável **maior** e será impresso pela instrução na linha 19.

## 4.4 Resumo da Aula

Esta aula trouxe conceitos importantes sobre o uso de vetores, que são uma estrutura homogênea que permite o armazenamento de **n** valores com fácil acesso por meio de índices. Assim, foram apresentados conceitos e exemplos práticos na linguagem C que permitem implementar operações de atribuição de valores em um vetor, acesso à valores em um vetor e utilizar estruturas de iteração para facilitar a manipulação de vetores.

Para atribuir e acessar valores em um vetor, basta utilizar o índice da posição do vetor, que inicia com **0 (zero)** e termina com **n-1**, sendo **n** o tamanho do vetor. Cada acesso, seja atribuição ou leitura de um valor requer uma instrução do programa, o que poderia levar o programador a ter de escrever centenas de instruções, em alguns casos milhares, mas, felizmente as estruturas de iteração permitem eliminar essa necessidade tornando possível o acesso à vetores com milhares de posições em algumas poucas linhas de código.

Um ponto importante comentado nesta aula está relacionado ao acesso e a atribuição em vetores, deve-se ter o devido cuidado para não acessar áreas inválidas de um vetor, pois ao fazer isso, é possível que o programa acesse informações de outra variável ou mesmo de outro programa e essa ação pode desencadear vários erros na aplicação. Assim, é preciso estabelecer corretamente a faixa de índices compatível com o tamanho do vetor.

## 4.5 Exercícios da Aula

Os exercícios desta lista foram Adaptados de [Lopes e Garcia \(2002, p. 277-303\)](#).

1. Faça um programa em C que armazene 15 números inteiros em um vetor e depois permita que o usuário digite um número inteiro para ser buscado no vetor, se for encontrado o programa deve imprimir a posição desse número no vetor, caso contrário, deve imprimir a mensagem: "Nao encontrado!".
2. Faça um programa que armazene 10 letras em um vetor e imprima uma listagem numerada.
3. Construa uma programa em C que armazene 15 números em um vetor e imprima uma listagem numerada contendo o número e uma das mensagens: par ou ímpar.
4. Faça um programa que armazene 8 números em um vetor e imprima todos os números. Ao final, imprima o total de números múltiplos de seis.
5. Faça um programa que armazene as notas das provas 1 e 2 de 15 alunos. Calcule e armazene a média arredondada. Armazene também a situação do aluno: 1-Aprovado ou 2-Reprovado. Ao final o programa deve imprimir uma listagem contendo as notas, a média e a situação de cada aluno em formato tabulado. Utilize quantos vetores forem necessários para armazenar os dados.
6. Construa um programa que permita armazenar o salário de 20 pessoas. Calcular e armazenar o novo salário sabendo-se que o reajuste foi de 8%. Imprimir uma listagem numerada com o salário e o novo salário. Declare quantos vetores forem necessários.
7. Crie um programa que leia o preço de compra e o preço de venda de 100 mercadorias (utilize vetores). Ao final, o programa deverá imprimir quantas mercadorias proporcionam:
  - lucro < 10%
  - 10% <= lucro <= 20%
  - lucro > 20%
8. Construa um programa que armazene o código, a quantidade, o valor de compra e o valor de venda de 30 produtos. A listagem pode ser de todos os produtos ou somente de um ao se digitar o código.
9. Faça um programa em C que leia dois conjuntos de números inteiros, tendo cada um 10 elementos. Ao final o programa deve listar os elementos comuns aos conjuntos.
10. Faça um programa que leia um vetor **vet** de 10 elementos e obtenha um vetor **w** cujos componentes são os fatoriais dos respectivos componentes de **vet**.
11. Construa um programa que leia dados para um vetor de 100 elementos inteiros. Imprimir o maior e o menor, sem ordenar, o percentual de números pares e a média dos elementos do vetor.

12. Crie um programa para gerenciar um sistema de reservas de mesas em uma casa de espetáculo. A casa possui 30 mesas de 5 lugares cada. O programa deverá permitir que o usuário escolha o código de uma mesa (100 a 129) e forneça a quantidade de lugares desejados. O programa deverá informar se foi possível realizar a reserva e atualizar a reserva. Se não for possível, o programa deverá emitir uma mensagem. O programa deve terminar quando o usuário digitar o código **0 (zero)** para uma mesa ou quando todos os 150 lugares estiverem ocupados.
13. Construa um programa que realize as reservas de passagens aéreas de uma companhia. O programa deve permitir cadastrar o número de 10 voos e definir a quantidade de lugares disponíveis para cada um. Após o cadastro, leia vários pedidos de reserva, constituídos do número da carteira de identidade do cliente e do número do voo desejado. Para cada cliente, verificar se há possibilidade no voo desejado. Em caso afirmativo, imprimir o número da identidade do cliente e o número do voo, atualizando o número de lugares disponíveis. Caso contrário, avisar ao cliente a inexistência de lugares. A leitura do número 0 (zero) para o voo desejado indica o término da leitura de reservas.
14. Faça um programa que armazene 50 números inteiros em um vetor. O programa deve gerar e imprimir um segundo vetor em que cada elemento é o quadrado do elemento do primeiro vetor.
15. Faça um programa que leia e armazene vários números, até digitar o número 0. Imprimir quantos números iguais ao último número foram lidos. O limite de números é 100.
16. Crie um programa em C para ler um conjunto de 100 números reais e informe:
  - quantos números lidos são iguais a 30
  - quantos são maior que a média
  - quantos são iguais a média
17. Faça um programa que leia um conjunto de 30 valores inteiros, armazene-os em um vetor e os imprima ao contrário da ordem de leitura.
18. Faça um programa em C que permita entrar com dados para um vetor VET do tipo inteiro com 20 posições, em que podem existir vários elementos repetidos. Gere um vetor VET2 ordenado a partir do vetor VET e que terá apenas os elementos não repetidos.
19. Suponha dois vetores de 30 elementos cada, contendo: código e telefone. Faça um programa que permita buscar pelo código e imprimir o telefone.
20. Faça um programa que leia a matrícula e a média de 100 alunos. Ordene da maior para a menor nota e imprima uma relação contendo todas as matrículas e médias.



# AULA 5

## Matrizes

### Metas da Aula

1. Entender e praticar os conceitos do uso de matrizes na linguagem C.
2. Aplicar variadas situações relacionadas ao uso de matrizes em programação.
3. Escrever programas que farão uso de matrizes.

### Ao término desta aula, você será capaz de:

1. Declarar uma matriz.
2. Obter os valores de variadas posições de uma matriz.
3. Atribuir valores em variadas posições da matriz.
4. Utilizar estrutura de iteração com matrizes.

## 5.1 Matrizes

Embora o uso de matrizes seja muito similar ao uso de vetores, é comum que os alunos encontrem especial dificuldade em compreender a manipulação de matrizes. Primeiro, é preciso então entender bem as matrizes para aplicar o seu uso na programação em linguagem C. Para iniciar, veja a figura 16, note que as matrizes estão dispostas em linhas e colunas, a matriz **A** tem 2 colunas e 2 linhas, a matriz **B** tem 2 colunas e 3 linhas e a **C** tem 3 colunas e 3 linhas. Se considerar **m** como sendo o número de linhas e **n** como sendo o número de colunas, então pode-se dizer também que a matriz **A** é de ordem 2 X 2, a matriz **B** é de ordem 3 X 2 e a matriz **C** é de ordem 3 X 3.

$$A = \begin{pmatrix} 3 & 7 \\ 8 & 3 \end{pmatrix} \quad B = \begin{pmatrix} 4 & 8 \\ 3 & 5 \\ 2 & 1 \end{pmatrix} \quad C = \begin{pmatrix} 0 & 4 & 3 \\ 7 & 2 & 1 \\ 1 & 1 & 2 \end{pmatrix}$$

Figura 16 – Exemplos de matrizes

Quando o número de colunas de uma matriz é igual ao número de linhas quer dizer que a matriz é quadrada, se o número de colunas é diferente do número de linhas, então a matriz é retangular. Na prática isso pode ser importante ou não no contexto da computação, pois se o problema a ser resolvido requer o cálculo matemático entre matrizes, então fará toda a diferença, pois as matrizes possuem várias propriedades que estão relacionadas ao seu formato (LOPES; GARCIA, 2002, p. 333). No mundo real, é comum lidar com matrizes, as vezes sem perceber, por exemplo, lembra do seu boletim de escola? E da planilha de despesas que você elaborou no Microsoft Excel para controlar seus gastos? Esses são exemplos de tabelas que possuem o formato de linhas e colunas de uma matriz.

O exemplo do boletim escolar na figura 17 é uma tabela de 5 linhas por 5 colunas, ou seja, uma matriz de ordem 5 X 5, e o exemplo da planilha de despesas apresentado na figura 18 é uma tabela de 5 linhas por 3 colunas, portanto, uma matriz de 5 X 3. Naturalmente nem todas as informações nestas matrizes são numéricas, mas, esta é uma

Disciplina	Prova 1	Prova 2	Nota Final	Situação
Matemática	4	3,7	7,7	Aprovado
História	5	3,5	8,5	Aprovado
Física	3	2	5	Recuperação
Geografia	4	3	7	Aprovado

Figura 17 – Exemplo 1 de tabela

Despesa	Data	Valor
Aluguel	10/05/2016	R\$800,00
Compra supermercado	15/05/2016	R\$332,00
Compra na farmácia	16/05/2016	R\$26,00
Táxi	16/05/2016	R\$23,50

Figura 18 – Exemplo 2 de tabela

realidade corriqueira nos problemas computacionais, pois os sistemas de informação, em geral, requerem o armazenamento de informações textuais, como nome, endereço, telefone, entre outras.

Dentre os exercícios da aula 4<sup>1</sup>, podem ser encontrados vários em que foi necessário armazenar vários dados, como dados de alunos ou dados de produtos. Para resolver estes problemas com o conhecimento adquirido até a aula 4, foi necessário utilizar vários vetores, sendo um vetor para armazenar cada informação, exemplo, para armazenar a matrícula, nome e nota final de vários alunos, é necessário declarar 3 vetores, um para cada informações dos  $n$  alunos. Contudo, o uso de matriz nos permite facilitar essa operação, uma vez que para a matriz é possível estabelecer mais de uma dimensão (ASCENCIO; CAMPOS, 2002; EDELWEISS; LIVI, 2014; SILVA; OLIVEIRA, 2014), assim, pode-se até dizer que uma matriz é um conjunto de vetores.

## 5.2 Dimensionando uma Matriz

Declarar uma matriz é similar à declarar um vetor e como a matriz possui mais de uma dimensão, então para dimensioná-la será necessário informar as dimensões extras. Veja a seguir a sintaxe para declarar a matriz (MIZRAHI, 2008; GOOKIN, 2016).

```
1 tipoMatriz nomeMatriz[Dimensao 1][Dimensao 2]...[Dimensao N];
```

Para definir o tipo da matriz utilize como referência a tabela 1. Para definir o nome da matriz, também siga as regras para definição de nomes de variáveis, assim, como os vetores. Exemplos de definição de nomes são apresentados na tabela 2. Como pode ver na sintaxe, uma matriz pode ter  $n$  dimensões, ou seja, embora tenha citado exemplos mais comuns como: 2 X 3 ou 4 X 3, entre outros, uma matriz pode ter 3, 4, 5...N dimensões, como: 3 X 4 X 5 ou 5 X 3 X 2 X 4 (MIZRAHI, 2008; BACKES, 2013).<sup>2</sup> Outro fator que influência no dimensionamento da matriz é que ela é uma estrutura homogênea, assim como o vetor, ou seja, mesmo que seja possível ter várias colunas e várias linhas, não será possível definir uma matriz com colunas de tipos diferentes (GOOKIN, 2016). Veja a seguir alguns exemplos de declaração.

```
1 //20 refere-se ao numero de linhas e 5 o numero de colunas
2 float notas[20][5];
3 int dados[50][3];
4 //30 refere-se ao numero de linhas e 30 ao numero de caracteres
5 char nomes[30][30];
```

No primeiro exemplo, na linha 2, foi declarada uma matriz com o nome **notas** do tipo **float** com duas dimensões, sendo 20 X 5, ou seja, 20 linhas por 5 colunas. No segundo exemplo, na linha 3, foi declarada uma matriz com o nome **dados** com dimensão de 50 X 3 e o terceiro exemplo, na linha 5, refere-se a uma matriz para armazenar nomes, que do ponto de vista do programador funcionará como se fosse um vetor, pois o primeiro argumento indica que serão 30 linhas, e o segundo argumento indica que para cada linha poderá ser armazenado um texto com até 30 caracteres, contudo, tecnicamente essa é uma matriz.

<sup>1</sup> As respostas para os exercícios da aula 4 estão no apêndice D.

<sup>2</sup> Nesta aula serão abordados apenas exemplos e exercícios com duas dimensões, pois foge ao escopo da aula um número maior que 2 dimensões.

### 5.3 Atribuição e obtenção de valores em matriz

A atribuição e obtenção de valores em matriz segue os mesmos procedimentos realizados para um vetor, a única diferença será que, neste caso, é preciso informar as dimensões no qual se deseja armazenar ou obter um valor. A figura 19 esclarece bem em como definir os argumentos de linha e coluna ao fazer acesso ou atribuição à uma matriz.

		matNotas			
0	4	2	3	9	<code>matNotas[1][1]</code>
1	1	3	3	7	<code>matNotas[2][3]</code>
2	3	2	1	6	
3	4	3	3	10	
	0	1	2	3	

Figura 19 – Exemplo de acesso à matriz

Como pode ver na figura 19, para acessar, por exemplo, o valor que intuitivamente pode-se dizer que seria a segunda nota do segundo aluno, então basta informar **1** para o primeiro argumento, a linha, e **1** para o segundo argumento, a coluna. Pois, apesar de ser a segunda nota do segundo aluno, as posições da matriz iniciam com o valor **0** (**zero**), assim como acontece com o vetor. Veja a seguir alguns exemplos de atribuição e acesso à valores de uma matriz.

```

1 //atribuindo valores a matriz matNotas
2 matNotas[1][2] = 2.5;
3 matNotas[2][2] = 3;
4
5 //atribuindo valor com o scanf
6 printf("Informe a segunda nota do aluno \n");
7 scanf("%f", &matNotas[1][1]);
8
9 //atribuindo valor com calculo de soma
10 matNotas[1][3] = matNotas[1][0] + matNotas[1][1] + matNotas[1][2];
11
12 //imprimindo a partir de uma posicao de matNotas
13 printf("Nota Final: %f \n", matNotas[1][3]);

```

As linhas 2 e 3 são exemplos de atribuição de valor à posições da matriz **matNotas**, na linha 7 é feita atribuição utilizando o **scanf**, a linha 10 mostra um exemplo de atribuição com um cálculo de soma, cujos valores obtidos são da própria **matNotas**. A linha 13 apresenta outro exemplo de obtenção de um valor da matriz, neste caso, para impressão na tela com a função **printf**.

## 5.4 Atribuição e acesso a valores com estrutura de iteração

É possível utilizar laços com matrizes também, contudo, no caso da matriz, em geral é necessário 1 laço para cada dimensão. Considerando o exemplo da figura 19, são necessários dois laços, uma vez que a matriz tem duas dimensões, sendo 4 linhas e 4 colunas. A seguir um exemplo em forma de exercício.

### 5.4.1 Exercício de Exemplo

Faça um programa em C que armazene as 3 notas de 4 alunos. Após a leitura o programa deve calcular a nota final com o somatório das três notas. Ao final deve imprimir as notas e a nota final para cada aluno.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 void main() {
4     float matNotas[4][4];
5     int i, j;
6     for (i=0; i<4; i++) {
7         for (j=0; j<3; j++) {
8             printf("Para o aluno %d informe a nota %d: \n", i+1, j+1);
9             scanf("%f", &matNotas[i][j]);
10        }
11        matNotas[i][3] = matNotas[i][0] + matNotas[i][1] + matNotas[i][2];
12    }
13
14    //imprimindo as notas dos alunos
15    for (i=0; i<4; i++) {
16        printf("Notas do Aluno %d\n", i);
17        for (j=0; j<3; j++) {
18            if (j != 2)
19                printf("%f \t", matNotas[i][j]);
20            else
21                printf("%f \n", matNotas[i][j]);
22        }
23        printf("Nota final.....: %f \n", matNotas[i][3]);
24    }
25 }
```

Na linha 4 foi feita a declaração da matriz **matNotas** com duas dimensões sendo 4 linhas por 4 colunas. Foi definida com esse tamanho para permitir armazenar para os quatro alunos, as 3 notas e a nota final. Na linha 5 foram declaradas duas variáveis, **i** e **j**, para uso como contador dos dois laços necessários para percorrer as duas dimensões da matriz. Entre as linhas 6 e 12 foi implementada a leitura das notas e o cálculo da nota final, note que o primeiro laço na linha 6 é responsável pela iteração das linhas, ou seja, a cada iteração deste laço uma linha da matriz é lida, começando da primeira. Na linha 7, o laço é responsável por percorrer as colunas, assim, a cada linha que é percorrida, todas as colunas também o são, e a cada coluna percorrida, uma nota é lida, com o **scanf** na linha 9.

Agora veja o **printf** da linha 8, percebeu que foram utilizados os contadores **i** e **j** para indicar para qual aluno são lançadas as notas no momento e qual nota está sendo lançada? Entendeu porque foi somado 1 no contador, como em **i+1**? Bem, a soma foi

realizada porque como os contadores iniciam com zero, seria estranho dizer que estão sendo lançadas as notas do aluno 0 (zero) ou que está sendo lançada a nota da avaliação 0, assim, bastou somar 1 para remover a estranheza.

Na linha 11 foi realizado o cálculo da nota final, este é mais um ponto de atenção, veja que o cálculo foi realizado dentro do escopo do primeiro laço, pois a chave, }, na linha 10 encerra o escopo do laço que percorre as colunas. Para entender porque, basta pensar o seguinte, para calcular a nota final é necessário ter todas as notas lançadas, assim, o laço responsável pelo lançamento das notas já deve estar completo. Por outro lado, é preciso calcular a nota para todos os alunos, portanto esse é o motivo do cálculo ter sido posicionado ao final do escopo do laço que percorre as linhas, ou seja, os alunos.

Entre as linhas 15 e 24 é realizada a impressão das notas dos alunos. Novamente o primeiro laço na linha 15 é responsável por percorrer as linhas da matriz e o laço na linha 17 é responsável por percorrer as colunas imprimindo as notas. Veja que o laço na linha 17 percorre até a terceira nota,  $j < 3$ , porque há a intenção de imprimir a nota final embaixo das notas individuais. Assim, é realizada a impressão das notas individuais nas linhas 19 ou 21, em que o **if** da linha 18 é responsável por determinar se chegou na última coluna da nota ou não, pois caso não tenha chegado na última coluna, então a impressão é acompanhada de uma tabulação conforme o **\t** na linha 19. Por fim, na linha 23 foi incluído o **printf** responsável pela impressão da nota final.

## 5.5 Resumo da Aula

Esta aula trouxe conceitos importantes sobre o uso de matrizes, que, assim como os vetores, são uma estrutura homogênea. Diferente dos vetores, as matrizes permitem o armazenamento em várias dimensões, sendo cada dimensão com  $n$  valores com fácil acesso por meio de índices. Assim, foram apresentados conceitos e exemplos práticos na linguagem C que permitem implementar as operações de atribuição de valores em uma matriz, acesso à valores em uma matriz e utilizar estruturas de iteração para facilitar a manipulação de matrizes.

Para atribuir e acessar valores em uma matriz, basta utilizar o índice da posição da matriz, que inicia com **0 (zero)** e termina com  **$n-1$** , sendo  $n$  o tamanho da dimensão da matriz, para cada dimensão, ou seja, uma matriz com duas dimensões, por exemplo, terá duas faixas de índice a ser informadas. Cada acesso, seja atribuição ou leitura de um valor requer uma instrução do programa, o que poderia levar o programador a ter de escrever centenas de instruções, em alguns casos milhares, mas, felizmente as estruturas de iteração permitem eliminar essa necessidade tornando possível o acesso à matrizes com milhares de posições em algumas poucas linhas de código. Contudo, para cada dimensão da matriz será necessário uma estrutura de iteração afim de percorrer todas as dimensões. Nesta aula, foram abordados apenas exemplos e exercícios com no máximo duas dimensões.

## 5.6 Exercícios da Aula

Os exercícios desta lista foram Adaptados de [Lopes e Garcia \(2002, p. 340-367\)](#).

1. Faça um programa em C que leia os elementos de uma matriz do tipo inteiro com tamanho 10 X 10. Ao final, imprima todos os elementos.
2. Faça um programa que leia os elementos de uma matriz do tipo inteiro com tamanho 3 X 3 e imprima os elementos multiplicando por 2.
3. Crie um programa que armazene dados inteiros em uma matriz de ordem 5 e imprima: Todos os elementos que se encontram em posições cuja linha mais coluna formam um número par.
4. Construa um programa que armazene dados em uma matriz de ordem 4 e imprima: Todos os elementos com números ímpares.
5. Faça um programa que permita entrar com valores em uma matriz **A** de tamanho 3 X 4. Gerar e imprimir uma matriz **B** que é o triplo da matriz **A**.
6. Crie um programa que leia valores inteiros em uma matriz A[2][2] e em uma matriz B[2][2]. Gerar e imprimir a matriz SOMA[2][2].
7. Construa um programa para ler valores para duas matrizes do tipo inteiro de ordem 3. Gerar e imprimir a matriz diferença.
8. Faça um programa que leia uma matriz 4 X 5 de inteiros, calcule e imprima a soma de todos os seus elementos.
9. Construa um programa em C que leia valores inteiros para a matriz  $A_{3 \times 5}$ . Gerar e imprimir a matriz SOMALINHA, em que cada elemento é a soma dos elementos de uma linha da matriz A. Faça o trecho que gera a matriz separado da entrada e da saída.
10. Construa um programa em C que leia valores inteiros para a matriz  $A_{3 \times 5}$ . Gerar e imprimir a matriz SOMACOLUNA, em que cada elemento é a soma dos elementos de uma coluna da matriz A. Faça o trecho que gera a matriz separado da entrada e da saída.
11. Entrar com valores para uma matriz  $C_{2 \times 3}$ . Gerar e imprimir a  $C^t$ .  
A matriz transposta é gerada trocando linha por coluna. Veja o exemplo a seguir:

$$C = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \quad C^t = \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix}$$

12. Uma floricultura conhecedora de sua clientela gostaria de fazer um programa que pudesse controlar sempre um estoque mínimo de determinadas plantas, pois todo dia, pela manhã, o dono faz novas aquisições. Criar um algoritmo que deixe cadastrar 50 plantas, nome, estoque mínimo, estoque atual. Imprimir ao final do programa uma lista das plantas que devem ser adquiridas.

13. A gerente do cabeleireiro Sempre Bela tem uma tabela em que registra as quantidades de serviços executados nos "pes", nas "mãos" e o serviço de podologia das cinco manicures. Sabendo-se que cada uma ganha 50% do que faturou ao mês, criar um programa que possa calcular e imprimir quanto cada uma vai receber, uma vez que não têm carteiras assinadas, os valores, respectivamente, são: R\$10,00, R\$15,00 e R\$30,00.
14. Crie um programa que leia e armazene os elementos de uma matriz inteira com tamanho 5 X 5 e imprimi-la. Troque, a seguir:
  - a segunda linha pela quinta;
  - a terceira coluna pela quinta;
  - a diagonal principal pela diagonal secundária.
15. A matriz dados contém na 1ª coluna a matrícula do aluno; na 2ª, o sexo (0 para feminino e 1 para masculino); na 3ª, o código do curso, e na 4ª, o CR (coeficiente de rendimento).  
Faça um programa que armazene esses dados sabendo-se que o código do curso é uma parte da matrícula: **aasccnnn** (**aa** ano, **s** semestre, **ccc** código do curso e **nnn** matrícula no curso).  
Além disso, um grupo empresarial resolveu premiar a aluna com CR mais alto de um curso cujo código deverá ser digitado. Suponha 10 alunos e que o CR é um nº inteiro.
16. Faça um programa em C que possa armazenar as alturas de dez atletas de cinco delegações que participarão dos jogos de verão. Imprimir a maior altura de cada delegação.
17. A Viação José Maria Rodrigues tem na Rodoviária de Rio Novo uma tabela contendo os horários de partidas dos ônibus para Juiz de Fora nos sete dias da semana. Faça um programa que possa armazenar esses horários e os horários do dia quando forem solicitados pelo funcionário, sabendo-se que, no máximo, são dez horários. Ao final, o programa deve imprimir a lista de horários para todos os dias.
18. Faça um programa que leia uma matriz 5 X 5 inteira e apresente uma determinada linha da matriz, solicitada via teclado.
19. Construa um programa que carregue uma matriz 12 X 4 com os valores das vendas de uma loja, em cada linha represente um mês do ano, e cada coluna, uma semana do mês. Calcule e imprima:
  - total vendido em cada mês do ano;
  - total vendido em cada semana durante todo o ano;
  - total vendido no ano.
20. Supondo que uma matriz apresente em cada linha o total de produtos vendidos ao mês por uma loja que trabalha com cinco tipos diferentes de produtos, construir um programa que leia esse total e, ao final, apresente o total de produtos vendidos em cada mês e o total de vendas por ano por produto.



# Tipos de Dados definidos pelo Programador

## Metas da Aula

1. Entender os conceitos necessários para definir um tipo em linguagem C.
2. Aprender a utilizar o novo tipo de dado definido.
3. Aplicar o uso de definição do próprio tipo de dado em variadas aplicações.
4. Aprender a escrever programas em linguagem C que utilizam tipos de dados definidos pelo programador.

## Ao término desta aula, você será capaz de:

1. Definir o próprio tipo de dado em linguagem C.
2. Determinar qual a melhor estrutura se aplica ao tipo de dado necessário ao problema.
3. Escrever programas que utilizam os tipos de dados definidos pelo programador.

## 6.1 Tipo de dado

Os tipos de dados aprendidos até o momento, como o **int** para armazenar números inteiros, o **float** para armazenar números reais, entre outros, conforme apresentado na tabela 1, são nativos da linguagem C e com limitações inerentes, visto serem tipos primitivos. Em geral a limitação está relacionada ao fato de que, com os tipos primitivos só é possível definir variáveis com dados homogêneos, por exemplo, com uma variável do tipo **int**, só é possível armazenar um número pertencente ao conjunto dos inteiros, com uma variável do tipo **char**, só é possível armazenar um caractere, desta forma, não existe, por exemplo, um tipo que permita armazenar um dado que seja composto por um tipo **int** em conjunto com um **char**.

Da mesma forma, temos outra limitação relacionada aos vetores e matrizes, que como já dito, são estruturas homogêneas, pois, nos exemplos vistos até agora, ao definir um vetor ou uma matriz, o tipo que cada célula recebe é primitivo e igual para todas as células. Ou seja, um vetor do tipo **int**, terá todas as células, ou seja, espaços de armazenamento, do tipo **int**, sendo assim uma estrutura homogênea. Contudo, no mundo real, são comuns os problemas que requerem o armazenamento e processamento de dados pertencentes ao mesmo contexto, mas que, tem natureza heterogênia. Exemplo, os dados de um cliente são algo comum, algo que toda empresa precisa armazenar e processar, agora imagine alguns dados básicos do cliente, como: nome (texto), CPF (conjunto de números inteiros combinado com caracteres especiais), data de nascimento (conjunto de números inteiros combinados com caracteres especiais), endereço (combinação de texto com números).

É fácil perceber que no mundo real frequentemente será necessário manipular dados heterogêneos, assim, as linguagens de programação, em geral, oferecem a possibilidade do programador definir os seus próprios tipos de dados, para que assim, possa confortavelmente resolver situações comuns no mundo real. O objetivo desta aula é explorar os conceitos necessários para dominar esta técnica. A seguir, entenda melhor sobre os registros ou estruturas de dados para avançar no conteúdo.

## 6.2 Estruturas de dados

Quando o programador necessita definir um tipo de dado heterogêneo, será necessário declarar uma estrutura de dados que é uma coleção de dados de quaisquer tipos, inclusive tipos já previamente definidos pelo programador, desta forma, é possível definir uma estrutura de dados heterogênea (BACKES, 2013, p. 145-146). Os dados que integram a estrutura são denominados de campos (EDELWEISS; LIVI, 2014, p. 294), conforme pode ser visto na figura 20. Note na figura 20, que o nome da estrutura é **funcionário** e os campos pertencentes à esta estrutura são: **cod**, **nome**, **salário**, **depto** e **cargo**. Naturalmente, cada campo tem um tipo específico, por exemplo, o campo **nome** armazena texto, já o campo **cod** armazena números, desta forma, esta estrutura **funcionário** é heterogênea.

Para entender bem o que são estes elementos, a estrutura de dados, o tipo de dado e a variável, basta fazer uma analogia, imagine que a estrutura seria o equivalente à planta de uma casa, e o tipo de dado seria uma planta padrão que seria utilizada para construir várias casas, e a variável definida a partir de uma estrutura ou um tipo de dado, seria a casa, ou seja, a estrutura de dados é a definição ou característica que terá o tipo de dado ou a variável, o tipo de dado é um padrão de estrutura e a variável é a referência para a memória alocada com as características do tipo definido pelo programador.

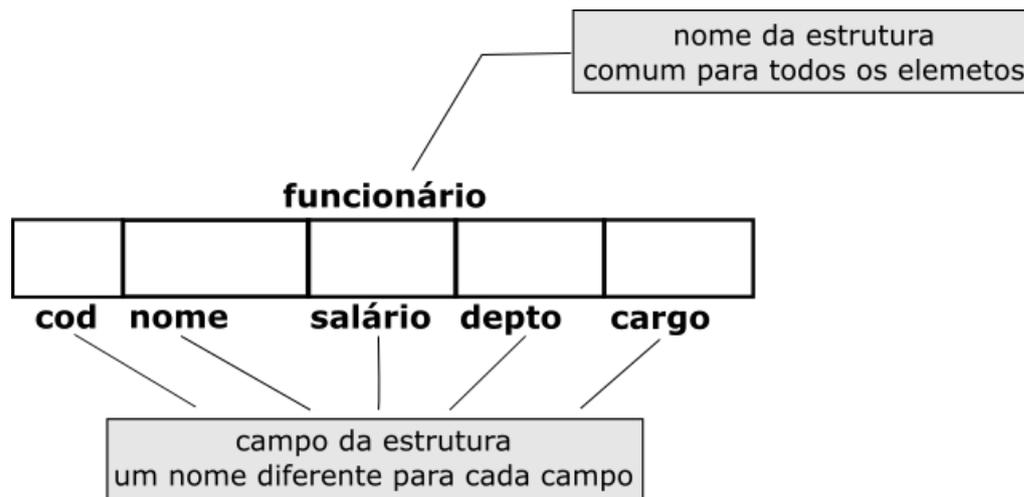


Figura 20 – Exemplo de estrutura

Fonte: Adaptado de (EDELWEISS; LIVI, 2014, p. 294)

## 6.3 Estruturas de dados em linguagem C

Como já foi mencionado, ao definir uma estrutura, na prática, é como se o programador definisse como será a característica do tipo de dado que ele pretende construir, desta forma, definir a estrutura não é definir o tipo de dado, contudo, a linguagem C nos permite utilizar a estrutura sem definir o tipo. Veja a seguir a sintaxe para definir uma estrutura em C.

```

1 //Sintaxe:
2 struct nome_estrutura {
3     tipo1 campo1;
4     tipo2 campo2;
5     ...
6     tipoN campoN;
7 };

```

Como pode ver na sintaxe, primeiro informa-se a palavra reservada **struct** para indicar o início de uma estrutura, na sequência, define-se o nome da estrutura que deve seguir o mesmo padrão de definição de nomes de variáveis, conforme aula 1. Os campos da estrutura são definidos entre chaves, conforme as linhas 2 e 7 da sintaxe, isso permite definir N campos para uma estrutura, no mínimo deve definir-se 1 campo e não há quantidade máxima. Cada campo é definido como se fosse uma variável, assim, informa-se o tipo e na sequência o nome, o tipo pode ser primitivo, conforme a tabela 1 ou um outro tipo definido pelo programador. Veja a seguir um exemplo de estrutura:

```

1 struct funcionario {
2     int cod;
3     char nome[30];
4     float salario;
5     int depto;
6     int cargo;
7 };

```

A estrutura de exemplo foi denominada de **funcionario** e os campos são **cod** do tipo **int**, **nome** do tipo **char** com tamanho 30, **salario** do tipo **float**, **depto** do tipo **int** e **cargo** do tipo **int**. Cada tipo e tamanho definido vai variar de acordo com a necessidade do problema, neste caso, por exemplo, **depto** e **cargo** foram definidos com o tipo inteiro, pois provavelmente pensou-se em armazenar apenas os códigos dessas informações, contudo, se a necessidade é a de armazenar apenas a descrição, então o tipo poderia ser **char**. Observe que neste exemplo foram definidos apenas campos com tipos nativos da linguagem C, mas ainda nesta aula serão apresentados os conceitos para definir campos de outras estruturas ou tipos customizados.

## 6.4 Variáveis e tipos de dados de estruturas

Após definir a estrutura, pode-se então utiliza-la como referência para definir uma variável ou um tipo de dado. Ambos os casos irão produzir "produtos" distintos, ao declarar uma variável de uma estrutura, é o mesmo que alocar espaço de memória com as características presentes na estrutura, por outro lado, ao definir um tipo de dado de uma estrutura, definiu-se um padrão a ser utilizado a partir daquela estrutura, desta forma, é como se estivesse dizendo: Esta estrutura será utilizada várias vezes! Assim, pode-se adotar como prática, que se a estrutura será utilizada poucas vezes, não há a necessidade de definir um tipo para ela, contudo, isso não é uma regra. Definir uma variável de uma estrutura é muito simples, basta substituir o tipo da variável pela palavra reservada **struct** seguido do nome da estrutura conforme o exemplo a seguir.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 struct funcionario {
5     int cod;
6     char nome[30];
7     float salario;
8     int depto;
9     int cargo;
10 };
11
12 void main()
13 {
14     struct funcionario func1, func2;
15 }
```

A definição da estrutura foi propositadamente repetida no exemplo, pois assim será mais claro o entendimento, note que o nome da estrutura na linha 4, **funcionario**, é idêntico ao nome da estrutura na definição da variável, na linha 14, isso permitirá à linguagem C determinar qual estrutura é desejada na definição da variável. Note também que foram declaradas duas variáveis para a mesma estrutura, no caso, **func1** e **func2**, o formato de declaração é o mesmo que a declaração de variáveis apresentado na aula 1, assim, basta separar por ",", quando desejar declarar mais de uma variável para o mesmo tipo.

Após declarada a variável, a forma de uso é similar à de uma variável tradicional, a única diferença é que deve-se indicar o nome da variável e o campo, no qual, deseja-se armazenar o dado. Um ponto importante é que não é necessário que todos os campos da variável sejam preenchidos. O exemplo já apresentado foi complementado para mostrar como atribuir dados à variáveis de estrutura. Veja a seguir:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 struct funcionario {
6     int cod;
7     char nome[30];
8     float salario;
9     int depto;
10    int cargo;
11 };
12
13 void main()
14 {
15     struct funcionario func1, func2;
16
17     //atribuindo dados a func1
18     func1.cod = 1;
19     strcpy(func1.nome, "Joao");
20     func1.salario = 1200;
21     //imprimindo os dados de func1
22     printf("Codigo: %d \n", func1.cod);
23     printf("Nome: %s \n", func1.nome);
24     printf("Salario: %f \n", func1.salario);
25
26     //atribuindo valores a func2 com o uso do scanf
27     printf("Informe o codigo: \n");
28     scanf("%d", &func2.cod);
29     printf("Informe o nome: \n");
30     scanf("%s", &func2.nome);
31     printf("Informe o salario: \n");
32     scanf("%f", &func2.salario);
33     //imprimindo os dados de func2
34     printf("Codigo: %d \n", func2.cod);
35     printf("Nome: %s \n", func2.nome);
36     printf("Salario: %f \n", func2.salario);
37 }
```

Como pode ver, até a linha 15 o programa é igual ao anterior, agora note a linha 18, veja que é uma atribuição direto no código-fonte, e como já mencionado foi indicado a variável, no caso **func1** e o campo, no caso **cod**, ambos foram associados por um ponto, ".", é assim que deve-se indicar o campo presente na variável, veja agora a linha 22, notou? Neste caso, foi impresso o valor do campo **cod** da variável **func1**. Mas, e como seria o uso do **scanf**? Não muda nada, a não ser o fato de que deve ser indicado o campo da variável, veja a linha 28, foi utilizado o **scanf** para ler um valor e armazenar no campo **cod** da variável **func2**. Assim, basicamente, o que muda ao fazer atribuições ou obter valor de variáveis com campos, é o fato de que o campo deve ser informado.

Podemos utilizar a estrutura para definir "**um padrão**", neste caso, definir um tipo de dado a partir da estrutura, para fazer isso em linguagem C, utilizamos o operador **typedef**, que permite definir tipos com base em outros tipos. Veja a seguir a sintaxe:

```
1 //Sintaxe:
2 typedef tipo_existente novo_nome;
```

Com o **typedef** pode-se por exemplo criar um tipo básico apenas mudando o nome, por exemplo, suponha que você, por algum motivo gostaria de indicar os tipos das variáveis com nomes em português, ao invés das abreviações em inglês nativas do C, então você poderia fazer algo como a seguir:

```
1 typedef int inteiro;
2 typedef float real;
3 typedef char caractere;
```

Notou no exemplo? Foram criados os tipos **inteiro**, **real** e **caractere** a partir dos tipos já existentes. Ao fazer isso, você pode utilizar os novos tipos definidos da mesma forma que aprendeu na aula 1. Mas, o nosso interesse não é esse, queremos utilizar o **typedef** para criar tipos heterogêneos. Desta forma, podemos ter um tipo de dado, padrão, definido para utilizar em nosso programa. Para definir funciona da mesma forma, substituindo o tipo pela palavra reservada **struct** acompanhado do nome da estrutura. Veja o exemplo a seguir.

```
1 struct funcionario {
2     int cod;
3     char nome[30];
4     float salario;
5     int depto;
6     int cargo;
7 };
8
9 typedef struct funcionario Funcionario;
```

Entre as linhas 1 e 7 foi definida a estrutura, cujo nome é **funcionario**, conforme a linha 1, na linha 9 foi definido o novo tipo, cujo nome é **Funcionario**, notou a diferença? Isso mesmo, como a linguagem C é *case sensitive*, utilizou deste fato para utilizar o mesmo nome da **struct** no **typedef**, alternando apenas a primeira letra para maiúsculo. Contudo, poderia ter sido utilizado qualquer nome para o **typedef**, desde que, siga as mesmas regras da definição do nome de variáveis.

Após definir um **typedef**, para utilizar basta seguir a mesma sintaxe na definição de variáveis e para utilizar a variável, utiliza-se o mesmo padrão adotado para variável de **struct**, para atribuição e obtenção de valor. Veja a seguir o exemplo adaptado.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 struct funcionario {
6     int cod;
7     char nome[30];
8     float salario;
9     int depto;
10    int cargo;
11 };
12
13 typedef struct funcionario Funcionario;
14
15 void main()
16 {
17     Funcionario func1, func2;
18 }
```

```
19 //atribuindo dados a func1
20 func1.cod = 1;
21 strcpy(func1.nome, "Joao");
22 func1.salario = 1200;
23 //imprimindo os dados de func1
24 printf("Codigo: %d \n", func1.cod);
25 printf("Nome: %s \n", func1.nome);
26 printf("Salario: %f \n", func1.salario);
27
28 //atribuindo valores a func2 com o uso do scanf
29 printf("Informe o codigo: \n");
30 scanf("%d", &func2.cod);
31 printf("Informe o nome: \n");
32 scanf("%s", &func2.nome);
33 printf("Informe o salario: \n");
34 scanf("%f", &func2.salario);
35 //imprimindo os dados de func2
36 printf("Codigo: %d \n", func2.cod);
37 printf("Nome: %s \n", func2.nome);
38 printf("Salario: %f \n", func2.salario);
39 }
```

Como pode ver no exemplo, entre as linhas 5 e 11 foi definida a estrutura **funcionario**, na linha 13 foi definido o tipo de dado **Funcionario** para a estrutura, na linha 17, as variáveis **func1** e **func2** foram declaradas com o novo tipo criado, **Funcionario**, note que neste caso, como é um tipo, não deve-se utilizar a palavra **struct**, nem mesmo **typedef**, contudo o nome do tipo criado deve ser exatamente igual para que o compilador consiga determinar quem é o novo tipo. Após declarar as duas variáveis, as linhas seguintes são exatamente iguais às do exemplo anterior, pois o formato de uso é idêntico.

## 6.5 Vetores de estruturas e tipos de dados

Ao utilizar as variáveis, mesmo sendo do tipo estrutura, há a limitação na quantidade de elementos a armazenar, exemplo, se declarar uma variável do tipo estrutura de **funcionario**, será possível armazenar vários dados do funcionário, mas apenas para 1 funcionário. Se é necessário armazenar mais de um funcionário, então é preciso utilizar vetores. Os vetores combinados com estrutura de dados heterogênea trazem muitos benefícios, pois ao definir um vetor de uma estrutura ou um tipo de dado, tiramos a limitação dele relacionada ao fato de que é uma estrutura homogênea e o mesmo assume as características de uma matriz heterogênea. Se ainda não consegue perceber isso com clareza, veja a figura 21.

Note que no exemplo apresentado na figura, primeiro foi definida a estrutura com nome **funcionario**, depois foi definido vetor **vetFunc** do tipo **struct funcionario** e o resultado do vetor é apresentado na parte inferior da figura 21, note que cada célula do vetor possui uma estrutura do tipo **funcionario**, que por sua vez comporta áreas distintas de armazenamento com tipos distintos, sendo assim, heterogênea, desta forma, o vetor passa a ter as características de uma matriz heterogênea.

Para declarar um vetor do tipo de uma estrutura basta seguir a mesma regra de declaração de vetores considerando que, ao invés de indicar um tipo nativo do C, indique o tipo definido pelo programador ou a estrutura, exemplo, ao invés de indicar **int**, indique **struct funcionario**. Veja a seguir um exemplo.

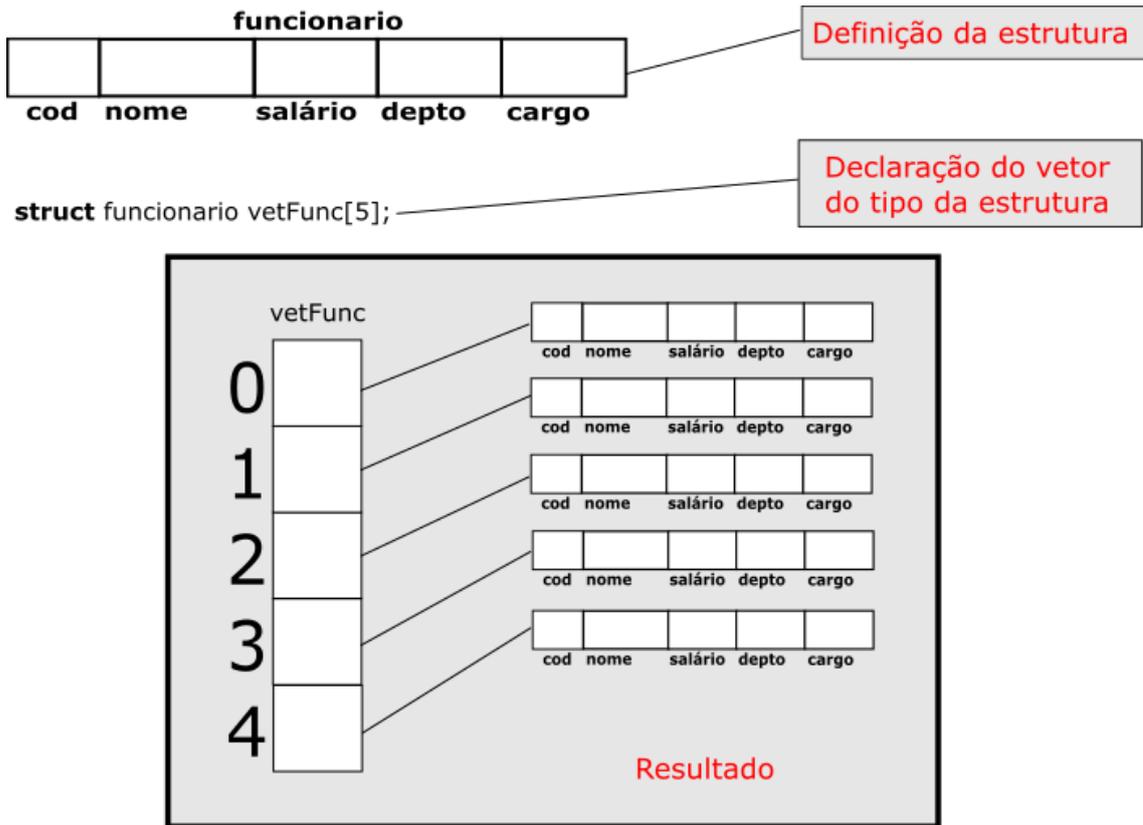


Figura 21 – Exemplo de vetor de estrutura

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 struct funcionario {
5     int cod;
6     char nome[30];
7     float salario;
8     int depto;
9     int cargo;
10 };
11
12 typedef struct funcionario Funcionario;
13
14 void main()
15 {
16     struct funcionario func1[5];
17     Funcionario func2[10];
18 }

```

No exemplo, veja que entre as linhas 4 e 10 foi definida a estrutura **funcionario**, na linha 12 foi definido o tipo **Funcionario**, na linha 16 foi declarado um vetor **func1** com tamanho 5 do tipo **struct funcionario** e na linha 17 foi declarado outro vetor **func2** com tamanho 10 e tipo **Funcionario**. Veja que é possível declarar o vetor tanto da estrutura quanto do novo tipo. A forma de uso da estrutura com vetor segue o mesmo padrão

adotado para as variáveis, contudo, deve-se indicar o índice. Veja a seguir o exemplo.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 struct funcionario {
5     int cod;
6     char nome[30];
7     float salario;
8     int depto;
9     int cargo;
10 };
11
12 typedef struct funcionario Funcionario;
13
14 void main()
15 {
16     Funcionario func[10];
17     int i;
18     for (i=0; i<10; i++) {
19         printf("Informe o codigo: \n");
20         scanf("%d", &func[i].cod);
21         printf("Informe o nome: \n");
22         scanf("%s", &func[i].nome);
23     }
24
25     for (i=0; i<10; i++) {
26         printf("Codigo: %d \n", func[i].cod);
27         printf("Nome: %s \n", func[i].nome);
28     }
29 }
```

Na linha 16 foi declarado o vetor de tamanho 10 e tipo **Funcionario**, entre as linhas 18 e 23 foi realizado um laço com **for** para ler os valores no vetor, para simplificar foram preenchidos apenas dois campos, o **cod** na linha 20 e o nome na linha 22, note que para atribuir o valor foi necessário indicar o nome do vetor seguido do índice e do campo. Entre as linhas 25 e 28 foi realizado outro laço **for**, neste caso para imprimir os dados lidos, novamente foi necessário indicar o nome do vetor, seguido do índice do campo a ser impresso, conforme pode ser visto nas linhas 26 e 27.

## 6.6 Estruturas aninhadas

No início da aula foi mencionado que é possível declarar na estrutura campos que são tipo de outras estruturas previamente definidas, quando um estrutura possui entre seus campos tipos pertencentes a outras estruturas, damos o nome de **estruturas aninhadas** (BACKES, 2013, p. 152). Veja a seguir um exemplo.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 struct departamento {
5     int cod;
6     char descricao[30];
7 };
```

```
8
9 struct cargo {
10     int cod;
11     char descricao[30];
12 };
13
14 struct funcionario {
15     int cod;
16     char nome[30];
17     float salario;
18     struct departamento depto;
19     struct cargo cargo;
20 };
21
22 typedef struct funcionario Funcionario;
23
24 void main()
25 {
26
27 }
```

Veja que interessante, entre as linhas 4 e 7 foi declarada a estrutura **departamento**, entre as linhas 9 e 12 foi declarada a estrutura **cargo**, entre as linhas 14 e 20 foi declarada a estrutura **funcionario**, mas note que o campo **depto** na linha 18, não é mais do tipo **int**, agora ele é do tipo **struct departamento**, da mesma forma foi feito com o campo **cargo**, na linha 19, desta forma, temos um aninhamento de estruturas. Pode-se utilizar também o tipo (**typedef**) ao invés de **struct**. A seguir o mesmo exemplo adaptado para **typedef**.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 typedef struct departamento {
5     int cod;
6     char descricao[30];
7 } Departamento;
8
9 typedef struct cargo {
10     int cod;
11     char descricao[30];
12 } Cargo;
13
14 typedef struct funcionario {
15     int cod;
16     char nome[30];
17     float salario;
18     Departamento depto;
19     Cargo cargo;
20 } Funcionario;
21
22 void main()
23 {
24
25 }
```

Dois pontos importantes são apresentados neste exemplo, primeiro note que, o **typedef** foi declarado em conjunto com o **struct**, tanto para a estrutura **departamento**, quanto para **cargo** e também para a estrutura **funcionario**, isso é útil apenas para resumir a codificação, mas não influi no desempenho da aplicação. O segundo ponto é a declaração dos campos **depto** e **cargo**, note que estes campos agora não são mais do tipo estrutura, mas dos tipos definidos **Departamento** e **Cargo**, respectivamente.

Desta forma, pode-se declarar campos de outros tipos definidos dentro de uma estrutura que também pode ser um tipo definido, além disso, não há uma limitação para o número de níveis na declaração aninhada, contudo, não seria prático incluir muitos níveis, pois isso irá complicar o entendimento para manutenção do código-fonte.

## 6.7 Resumo da Aula

Esta aula trouxe conceitos importantes sobre a declaração de tipos definidos pelo usuário com o uso de **struct** e **typedef**, destacou-se a importância desses tipos nos problemas computacionais tendo em vista que a grande maioria dos problemas requerem o uso de estruturas heterogêneas, o que não é possível com as variáveis e vetores declarados a partir dos tipos nativos da linguagem C.

Ao longo da aula foram apresentados os conceitos necessários para declarar e utilizar as estruturas de dados, em C **struct** e os tipos customizados, em C **typedef**. Ambos os recursos podem ser aplicados tanto com variáveis, bem como com vetores, o que traz um excelente benefício, pois o uso com vetores, permite obter uma estrutura similar à uma matriz heterogênea, o que não é possível com a declaração de uma matriz com tipos básicos.

## 6.8 Exercícios da Aula

Parte dos exercícios desta lista foram Adaptados de [Backes \(2013, p. 161-162\)](#).

1. Implemente um programa em C que leia o **nome**, a **idade** e o **endereço** de uma pessoa e armazene esses dados em uma estrutura. Em seguida, imprima na tela os dados da estrutura lida.
2. Crie uma estrutura para representar as coordenadas de um **ponto** no plano (posições X e Y). Em seguida, declare e leia do teclado um ponto e exiba a distância dele até a origem das coordenadas, isto é, posição (0, 0). Para realizar o cálculo, utilize a fórmula a seguir <sup>1</sup>:

$$d = \sqrt{(X_B - X_A)^2 + (Y_B - Y_A)^2} \quad (6.1)$$

Em que:

- d = distância entre os pontos A e B
  - X = coordenada X em um ponto
  - Y = coordenada Y em um ponto
3. Crie uma estrutura para representar as coordenadas de um **ponto** no plano (posições X e Y). Em seguida, declare e leia do teclado dois pontos e exiba a distância entre eles, considere a mesma fórmula do exercício anterior.
  4. Cria uma estrutura chamada **retângulo**. Essa estrutura deverá conter o ponto superior esquerdo e o ponto inferior direito do retângulo. Cada ponto é definido por uma estrutura **Ponto**, a qual contém as posições X e Y. Faça um programa que declare e leia uma estrutura **retângulo** e exiba a área e o comprimento da diagonal e o perímetro desse retângulo.
  5. Usando a estrutura **retângulo** do exercício anterior, faça um programa que declare e leia uma estrutura **retângulo** e um **ponto**, e informe se esse ponto está ou não dentro do retângulo.
  6. Crie uma estrutura representando um aluno de uma disciplina. Essa estrutura deve conter o número de matrícula do aluno, seu nome e as notas de três provas. Defina também um tipo para esta estrutura. Agora, escreva um programa que leia os dados de cinco alunos e os armazena nessa estrutura. Em seguida, exiba o nome e as notas do aluno que possui a maior média geral dentre os cinco.
  7. Crie uma estrutura representando uma hora. Essa estrutura deve conter os campos hora, minuto e segundo. Agora, escreva um programa que leia um vetor de cinco posições dessa estrutura e imprima a maior hora.
  8. Crie uma estrutura capaz de armazenar o nome e a data de nascimento de uma pessoa. Faça uso de estruturas aninhadas e definição de novo tipo de dado. Agora, escreva um programa que leia os dados de seis pessoas. Calcule e exiba os nomes da pessoa mais nova e da mais velha.

<sup>1</sup> Dica: Se tiver dificuldade com os cálculos de potência e raiz, consulte a aula 7 que trata deste assunto

9. Crie uma estrutura representando um atleta. Essa estrutura deve conter o nome do atleta, seu esporte, idade e altura. Agora, escreva um programa que leia os dados de cinco atletas. Calcule e exiba os nomes do atleta mais alto e do mais velho.
10. Usando a estrutura "atleta" do exercício anterior, escreva um programa que leia os dados de cinco atletas e os exiba por ordem de idade, do mais velho para o mais novo.
11. Escreva um programa que contenha uma estrutura representando uma data válida. Essa estrutura deve conter os campos dia, mês e ano. Em seguida, leia duas datas e armazene nessa estrutura. Calcule e exiba o número de dias que decorrem entre as duas datas.
12. Astolfolov Oliveirescu é técnico de um time da série C do poderoso campeonato de futebol profissional da Albânia. Ele deseja manter os dados dos seus jogadores guardados de forma minuciosa. Ajude-o fazendo um programa para armazenar os seguintes dados de cada jogador: nº da camisa, peso (kg), altura (m) e a posição em que joga (atacante, defensor ou meio campista). Lembre-se que o time tem 22 jogadores, entre reservas e titulares. Leia os dados e depois gere um relatório no vídeo, devidamente tabulado/formatado.
13. Um clube social com 37 associados deseja que você faça um programa para armazenar os dados cadastrais desses associados. Os dados são: nome, dia, mês e ano de nascimento, valor da mensalidade e quantidade de dependentes. O programa deverá ler os dados e imprimir depois na tela. Deverá também informar o associado (ou os associados) com o maior número de dependentes.
14. Crie um programa que tenha uma estrutura para armazenar o nome, a idade e número da carteira de sócio de 50 associados de um clube. Crie também uma estrutura, dentro desta anterior, chamada **dados** que contenha o endereço, telefone e data de nascimento.
15. Crie um programa com uma estrutura para simular uma agenda de telefone celular, com até 100 registros. Nessa agenda deve constar o nome, sobrenome, número de telefone móvel, número de telefone fixo e e-mail. O programa deverá fazer a leitura e, após isso, mostrar os dados na tela.

# Funções

## Metas da Aula

1. Entender e praticar os conceitos do uso de funções na linguagem C.
2. Aplicar variadas situações relacionadas ao uso de funções em programação.
3. Entender em como utilizar funções presentes em bibliotecas disponibilizadas com a linguagem C.
4. Aprender a escrever novas funções em linguagem C.

## Ao término desta aula, você será capaz de:

1. Definir uma função.
2. Utilizar uma função da linguagem C.
3. Escrever novas funções em linguagem C.

## 7.1 Funções

As funções estão presentes em todo o código de um programa, embora nas aulas anteriores elas não tenham sido mencionadas, vários programas construídos fizeram uso de funções. Exemplos de funções utilizadas, são: `scanf()`, `printf()`, `main()`, entre outras. As funções são blocos de código encapsulado e que podem ser reutilizados ao longo do programa (ANICHE, 2015), assim, a função `printf()`, por exemplo, trata-se de um bloco de instruções que é capaz de produzir a saída de uma informação para o usuário do programa. Como, em vários momentos do ciclo de vida do programa, se faz necessário apresentar uma informação por um dispositivo de saída, então é conveniente que as instruções responsáveis por esta ação, sejam encapsuladas em uma função, para que a ação possa ser invocada a qualquer momento no programa.

Desta forma, podemos entender que as funções são úteis quando identificamos trechos de código que, em geral, serão reutilizados em vários trechos do programa. Assim, o programador pode criar as suas próprias funções para reutilizar o código-fonte que julgar necessário (LAUREANO, 2005). Além disso, em geral, a linguagem C, está acompanhada de bibliotecas que trazem outras várias funções que facilitam a vida do programador, pois trazem a possibilidade de executar operações que não estão disponíveis nativamente na linguagem, por exemplo, não existe na linguagem C um operador para o cálculo de raiz quadrada ou de potência, desta forma, para fazer esses cálculos o programador deve utilizar apenas os operadores básicos apresentados na tabela 4, ou, pode utilizar uma função pronta que faz parte de uma biblioteca chamada *math*.

## 7.2 Funções presentes na linguagem C

Várias bibliotecas acompanham os ambientes de desenvolvimento em linguagem C, como é o caso do **CodeBlocks**, do **DevC++**, entre outros.<sup>1</sup> Uma dica que pode lhe poupar tempo é, sempre que se deparar com a necessidade de executar um conjunto de instruções para realizar uma operação que refere-se a uma necessidade comum de outros usuários, investigue se já existe uma função antes de tentar implementar a sua.

### 7.2.1 Funções matemáticas

A biblioteca **math** disponibiliza várias funções matemáticas, como funções trigonométricas para o cálculo do cosseno, função **cos**, função para cálculo de logaritmo, como a função **log**, entre outras, contudo, apenas duas funções serão discutidas, a função para cálculo de raiz quadrada, `sqrt()` e a função para cálculo de potência, `pow(x, y)`.

#### 7.2.1.1 Função para cálculo da raiz quadrada

Como já mencionado, não há um operador nativo na linguagem C que permita calcular a raiz quadrada de um número, contudo, podemos utilizar a função, `sqrt()`, disponibilizada pela biblioteca **math** para fazer este cálculo (SCHILDT, 1996). A seguir a sintaxe da função `sqrt()`.

```
1 //Sintaxe:
2 #include <math.h>
3 double sqrt(double num);
```

<sup>1</sup> Contudo nesta aula serão apresentadas apenas algumas funções que serão úteis nas aulas seguintes.

Conforme pode ser visto na sintaxe, a função retorna um **double**, que no caso será o resultado do cálculo da raiz quadrada do número que foi passado como argumento na invocação da função. O número passado como argumento também é do tipo **double**, contudo, tanto o retorno, quanto o argumento podem ser do tipo **float**, visto que um **double** pode assumir valores do tipo **float**. Veja a seguir um exemplo em forma de exercício.

## 7.2.2 Exercício de Exemplo

Faça um programa em C que leia um número e após a leitura, calcule e imprima a raiz quadrada deste número.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4
5 void main()
6 {
7     float numero, raiz;
8     printf("Informe um numero: \n");
9     scanf("%f", &numero);
10
11     raiz = sqrt(numero);
12
13     printf("A raiz quadrada do numero e: %f \n", raiz);
14 }
```

Na resposta proposta para o exercício foi adicionada a biblioteca **math** na linha 3, sempre que precisar utilizar uma função de alguma biblioteca, será necessário adicionar o arquivo cabeçalho correspondente no início do código. Na linha 7 foram declaradas as variáveis **numero**, para armazenar o número lido, e **raiz** para armazenar o resultado do cálculo da raiz quadrada. As linhas 8 e 9 são responsáveis pela leitura do número. A linha 11 é o ponto de interesse neste exercício, veja que a variável **raiz** recebe a função **sqrt(numero)**, isso, porque a função **sqrt()** retorna um valor e requer um argumento, desta forma, a função irá calcular a raiz quadrada do número passado como argumento e retornar à instrução que a invocou o resultado do cálculo, neste caso, a atribuição à variável **raiz** é que foi responsável pela invocação da função.

### 7.2.2.1 Função para cálculo da potência

A função para cálculo da potência, **pow()**, requer dois argumentos conforme pode ser visto na sintaxe a seguir. O primeiro argumento é a base, ou seja, o número que deseja-se que seja calculado a sua potência, e o segundo argumento é o expoente, que é o valor no qual, o número da base será elevado (SCHILDT, 1996). Assim como na função **sqrt()**, a função **pow()** também retorna um **double**, no caso, o valor calculado, e também admite valores do tipo **float**. Veja na sequência um exercício de exemplo do uso da função **pow()**.

```
1 //Sintaxe:
2 #include <math.h>
3 double pow(double base, double exp);
```

### 7.2.3 Exercício de Exemplo

Faça um programa em C que leia um número e um expoente, após a leitura, calcule e imprima a potência do número lido em relação ao expoente também lido.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4
5 void main()
6 {
7     float base, expoente, potencia;
8     printf("Informe a base: \n");
9     scanf("%f", &base);
10    printf("Informe o expoente: \n");
11    scanf("%f", &expoente);
12
13    potencia = pow(base, expoente);
14
15    printf("A potencia do numero e: %f \n", potencia);
16 }
```

Assim como no exemplo anterior, a biblioteca *math* foi incluída na linha 3. Na linha 7 foram declaradas as variáveis necessárias, **base**, para armazenar o número que será elevado a uma potência, **expoente**, que será utilizada para armazenar a potência desejada e **potencia** para armazenar o resultado do cálculo. Entre as linhas 8 e 11 os valores foram lidos e armazenados nas respectivas variáveis. Na linha 13 temos o uso da função **pow()**, veja que, o acionamento da função foi realizado pela uso da atribuição à variável **potencia**, e a função recebeu dois argumentos, o primeiro argumento é o valor da base e o segundo argumento é o valor do expoente. Com esses dois valores a função retornará o resultado do cálculo que será atribuído à variável **potencia**. Na linha 15 a função **printf()** se encarrega de apresentar o resultado do cálculo.

### 7.2.4 Funções de tratamento de texto

A linguagem C não dispõe, nativamente, do tipo **string**, como é comum em outras linguagens, desta forma, para armazenar texto, é preciso utilizar um vetor do tipo **char**, como se trata de um vetor, então, algumas operações não são possíveis de forma direta, como é possível com outros tipos nativos. Duas operações em especial são as de maior uso, atribuição e comparação. A seguir serão discutidas duas funções da biblioteca **string** que facilitam essas tarefas.

#### 7.2.4.1 Função para atribuição de texto

```
1 void main()
2 {
3     char nome[30], novoNome[30];
4     nome = novoNome;
5     .
6     .
7     .
```

Em linguagem C, a operação na linha 4 do código-fonte apresentado neste início de tópico **não** é permitida, porque como **nome** e **novoNome** são dois vetores, não há como garantir, por exemplo, que os dois vetores tenham a mesma dimensão, ou que as dimensões sejam compatíveis, então, não é possível fazer a atribuição direta, como na linha 4 do trecho de código à seguir. É necessário então, percorrer o vetor de origem e, a cada posição, atribuir o valor ao vetor de destino, contudo, várias validações devem ser previamente realizadas, como a compatibilidade de tamanho entre os vetores.

Realizar essas operações requer um bom trabalho, além disso, trata-se de uma operação recorrente em programação, desta forma, é conveniente ter uma função que facilite esta operação. A biblioteca **string** disponibiliza a função **strcpy()** que faz a atribuição de um vetor do tipo **char** a outro vetor, também do tipo **char**. Além disso, faz as verificações necessárias, lançando as exceções quando for o caso. A seguir a sintaxe da função e um exemplo de uso.

```
1 //Sintaxe:
2 #include <string.h>
3 char *strcpy(char *str1, const char *str2);
```

Como pode-se ver na sintaxe, a biblioteca **string.h** deve ser incluída no código-fonte. Outro detalhe é que a função **strcpy()** recebe dois parâmetros, o primeiro, **str1**, é o vetor que receberá os valores do segundo vetor, que no caso é o segundo argumento, **str2**, ambos os argumentos devem ser do tipo **char** (SCHILDT, 1996). Como a função recebe como argumento os dois vetores envolvidos na operação, então não há necessidade da atribuição tradicional, como pode ser visto no exemplo a seguir, na linha 11, a função é invocada com os dois argumentos necessários e ela se encarregará de fazer o restante do trabalho.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 void main()
6 {
7     char nome[30], novoNome[30];
8     printf("Informe o novo nome: \n");
9     scanf("%s", &novoNome);
10
11     strcpy(nome, novoNome);
12
13     printf("O novo nome e: %s \n", nome);
14 }
```

#### 7.2.4.2 Função para comparação de texto

Comparar texto de forma direta, por exemplo, utilizando o sinal de igualdade, é outra ação indisponível na linguagem C, pois, novamente, o uso de vetores, impede isso, visto que, comparar dois vetores, implica em analisar cada posição comparando com a posição correspondente do outro vetor e novamente, é uma operação recorrente que demanda uma função para isso, desta forma, a biblioteca **string** disponibiliza uma função para facilitar essa operação, **strcmp()**, que faz a comparação entre dois vetores devolvendo um inteiro **0 (zero)**, caso os dois textos sejam iguais e diferente de zero, no caso contrário (SCHILDT, 1996). Veja a seguir a sintaxe:

```

1 //Sintaxe:
2 #include <string.h>
3 int strcmp(const char *str1, const char *str2);

```

A biblioteca **string.h** deve ser incluída para uso da função **strcmp()**. Veja que a função retorna um inteiro, se o retorno é igual a **0 (zero)**, os dois textos são iguais, se é **diferente de 0 (zero)**, os dois textos são diferentes. O primeiro argumento informado é o primeiro vetor do tipo **char** que pretende-se comparar e o segundo argumento é o vetor a ser comparado com o primeiro. Veja a seguir um exemplo de uso da função.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 void main()
6 {
7     char nome[30]={'t','e','s','t','e'}, novoNome[30];
8     printf("Informe o novo nome: \n");
9     scanf("%s", &novoNome);
10
11     //if (nome == novoNome)  -- nao funciona!
12     if (strcmp(nome, novoNome) == 0)
13         printf("Os nomes sao iguais! \n");
14     else
15         printf("Os nomes sao diferentes! \n");
16 }

```

Na linha 3 do exemplo, a biblioteca **string.h** é incluída ao código-fonte. Veja o código comentado na linha 11, note que trata-se da comparação na forma tradicional, contudo esse código não funciona, pois **nome** e **novoNome** são vetores, assim, na linha 12 foi utilizada a função **strcmp()** para fazer a comparação, então o retorno da função é comparado com 0, se a condição retornar verdadeiro, a linha 13 será executada, se retornar falso, então a linha 15 será executada.

## 7.3 A forma geral de uma função

Como já mencionado, havendo uma situação em que é necessário reutilizar um código várias vezes, é conveniente que este código seja uma função. Em boa parte das situações, não haverá uma função pronta que atenda à necessidade em questão, neste caso, será necessário escrever a função. A linguagem C permite definir funções e para isso, basta entender como elas funcionam e como aplicar estes conceitos ao problema computacional que pretende-se resolver. Veja a seguir a sintaxe para definir uma função.

```

1 //Sintaxe:
2 tipoRetorno nomeFuncao(lista de parametros) {
3     corpo da funcao;
4     return valorRetornado;
5 }

```

Na linha 2 da sintaxe, veja que, primeiro é necessário definir o "**tipoRetorno**", o retorno é o dado de saída da função, após o processamento. Este tipo deve ser definido conforme a tabela 1 que apresenta os tipos básicos da linguagem C, assim, o retorno

pode ser definido, por exemplo, com o tipo **int** e neste caso, o retorno da função será um número pertencente ao conjunto dos inteiros. Contudo, há situações em que não é necessário que a função retorne um valor, neste caso, o retorno da função deve ser definido como **void**. É possível também, definir o retorno com um tipo definido pelo programador<sup>2</sup>. Ainda na linha 2, veja que deve-se dar um nome à função, neste caso, substitua "**nomeFuncao**", por um nome que seja condizente com a ação da função, ou seja, um nome intuitivo. Para definir o nome da função, utilize as mesmas regras, já apresentadas na aula 1, para definir o nome de variáveis.

A lista de parâmetros, ainda na linha 2, é o conjunto de variáveis que serão necessárias à função, para que a mesma, resolva o problema computacional em questão, ou seja, são os dados de entrada da função. Assim, pode-se definir quantos forem necessários os dados de entrada para que uma função resolva um problema. Por exemplo, imagine uma função para calcular a área de um retângulo, sabe-se que para fazer este cálculo é necessário a base e a altura do retângulo, assim, seria necessário definir estes dois parâmetros para esta função. Seguindo o mesmo exemplo, como o objetivo é calcular a área do retângulo, então é natural que o retorno dessa função seja o valor calculado, desta forma, pode-se concluir que o retorno deve ser do tipo **float**.

Na linha 3, "**corpo da funcao;**", deve ser escrito o código-fonte que se encarregará de resolver o problema para o qual a função está sendo criada. Naturalmente, o código-fonte não está limitado à 1 linha, podem ser utilizadas tantas quantas forem necessárias. O que define o início e término da função é a chave, {, na linha 2 e a chave, }, na linha 5. Na linha 4, temos o "**return valorRetornado;**", que será sempre obrigatório quando o retorno da função for diferente de **void**. A palavra **return** é fixa e a palavra **valorRetornado** deve ser substituída pelo valor que deseja-se retornar, que em geral, estará em uma variável. Veja a seguir um exercício para exemplificar.

### 7.3.1 Exercício de Exemplo

Faça um programa em C que calcule a área de um retângulo, para isso o programa deve ler a altura e a base. O cálculo deve ser feito em uma função. Após calcular o programa deve imprimir o valor da área do retângulo.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 float areaRetangulo(float base, float altura) {
5     float area = base * altura;
6     return area;
7 }
8
9 void main()
10 {
11     float vbase, valtura, varea;
12     printf("Informe a base do retangulo: \n");
13     scanf("%f", &vbase);
14     printf("Informe a altura do retangulo: \n");
15     scanf("%f", &valtura);
16     varea = areaRetangulo(vbase, valtura);
17     printf("A area do retangulo e: %f \n", varea);
18 }
```

<sup>2</sup> Para mais detalhes sobre como definir um tipo próprio, consulte a aula 6

Veja que o programa têm duas funções, a função de cálculo da área do retângulo está entre as linhas 4 e 7 e entre as linhas 9 e 18 está a função *main*, que é necessária em todos os programas. O código na função *main* é responsável pela execução da leitura dos valores da base, na variável *vbbase*, e da altura, na variável *valtura*, e a invocação da função *areaRetangulo*, na linha 16, por fim, a impressão da área resultante do cálculo na linha 17. Veja que a função é invocada na linha 16, por meio da atribuição do retorno da função à variável *varea*, e na invocação da função são informados os dois argumentos necessários ao cálculo, por meio das variáveis *vbbase* e *valtura*.

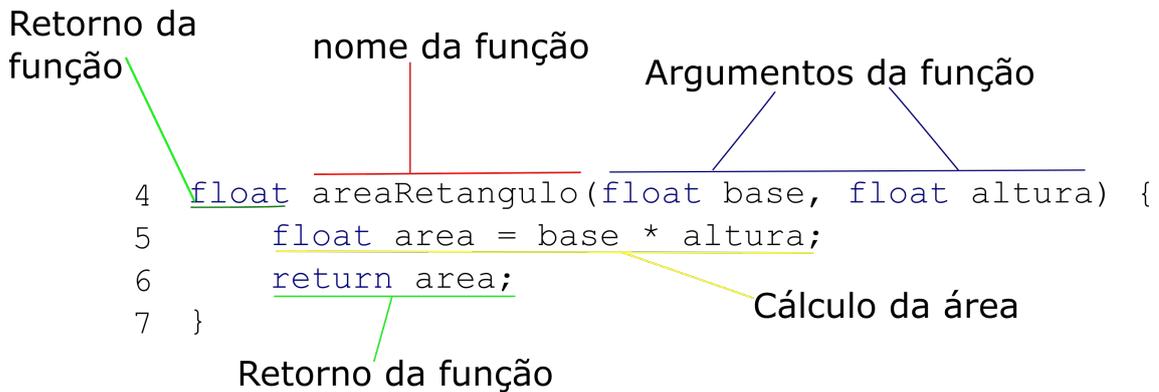


Figura 22 – Função de cálculo da área do retângulo

Veja na figura 22 o trecho do código-fonte da função, cada parte da função foi identificada. O nome da função é *areaRetangulo* e os argumentos da função são *base* e *altura* do tipo *float*. Após o cálculo ser realizado na linha 5, o retorno do valor calculado é feito na linha 6. Veja também que na linha 5, são utilizadas as variáveis do argumento da função para o cálculo, todos os argumentos presentes na função podem ser utilizados como variáveis no corpo da função, e de fato, o objetivo dos argumentos é este, servir de entrada para os dados a serem processados.

## 7.4 Protótipos de funções

No código de exemplo apresentado, observa-se que a função de cálculo da área do retângulo foi incluída antes da função *main*, esta é uma forma de disponibilizar uma função para uso no código da função principal, assim, visto que a linguagem C é estruturada, qualquer função com o código disponível antes da função *main*, estará disponível para uso na função *main*.

Contudo, imagine um arquivo de código-fonte com 20 ou 30 funções diferentes! O código-fonte certamente ficará um tanto confuso. Uma alternativa para a organização das funções em um código-fonte é o uso de protótipos, ao utiliza-los você terá uma lista das funções antes da função principal, *main* e os códigos das funções devem ser disponibilizados após a função *main*. Outra opção que trará melhor organização ao código-fonte é o uso de modularização<sup>3</sup>. A seguir, veja o exemplo anterior adaptado para o uso de protótipo.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3

```

<sup>3</sup> A modularização não faz parte do escopo deste livro

```
4 float areaRetangulo(float base, float altura);
5
6 void main()
7 {
8     float vbase, valtura, varea;
9     printf("Informe a base do retangulo: \n");
10    scanf("%f", &vbase);
11    printf("Informe a altura do retangulo: \n");
12    scanf("%f", &valtura);
13    varea = areaRetangulo(vbase, valtura);
14    printf("A area do retangulo e: %f \n", varea);
15 }
16
17 float areaRetangulo(float base, float altura) {
18     float area = base * altura;
19     return area;
20 }
```

Observe a linha 4, este é o protótipo, ou seja, o protótipo equivale ao cabeçalho da função, basta repetir o código da primeira linha da função e a função deve ser escrita abaixo do código-fonte da função principal, conforme as linhas 17 a 20. E caso seja necessário adicionar mais uma função neste código? Veja a seguir uma adaptação do mesmo exemplo, em que será adicionada a função para o cálculo da área do triângulo.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 float areaRetangulo(float base, float altura);
5 float areaTriangulo(float base, float altura);
6
7 void main()
8 {
9     float vbase, valtura, vareaRet, vareaTri;
10    printf("Informe a base do retangulo: \n");
11    scanf("%f", &vbase);
12    printf("Informe a altura do retangulo: \n");
13    scanf("%f", &valtura);
14    vareaRet = areaRetangulo(vbase, valtura);
15    vareaTri = areaTriangulo(vbase, valtura);
16
17    printf("A area do retangulo e: %f \n", vareaRet);
18    printf("A area do triangulo e: %f \n", vareaTri);
19 }
20
21 float areaRetangulo(float base, float altura) {
22     float area = base * altura;
23     return area;
24 }
25
26 float areaTriangulo(float base, float altura) {
27     float area = (base * altura) / 2;
28     return area;
29 }
```

Agora note que além do protótipo da função de cálculo da área do retângulo, na linha 4, foi adicionado também o protótipo do cálculo da área do triângulo, na linha 5. Na linha 9, foi adicionada uma variável para armazenar o resultado do cálculo da área do triângulo, e o nome da variável responsável por armazenar o resultado do cálculo da área do retângulo foi ajustada para manter o padrão na nomenclatura. Para ambos os cálculos são utilizados os mesmos valores, **base** e **altura**, assim, entre as linhas 10 e 13 não houve alterações. Na linha 15, foi adicionada a invocação para a função de cálculo da área do triângulo, note o nome da função para identificar as diferenças. Foi adicionada também a linha 18 para a impressão da área do triângulo.

A função de cálculo da área do triângulo foi escrita entre as linhas 26 e 29. Notou se há alguma semelhança com o cálculo da área do retângulo? Sim, há! Para calcular a área do triângulo, basta dividir o resultado do cálculo da área do retângulo por 2. Desta forma, podemos melhorar este código utilizando a função de cálculo da área do retângulo na função de cálculo da área do triângulo. Então podemos invocar uma função em outra função? Sim, veja a seguir os ajustes.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 float areaRetangulo(float base, float altura);
5 float areaTriangulo(float base, float altura);
6
7 void main()
8 {
9     float vbase, valtura, vareaRet, vareaTri;
10    printf("Informe a base do retangulo: \n");
11    scanf("%f", &vbase);
12    printf("Informe a altura do retangulo: \n");
13    scanf("%f", &valtura);
14    vareaRet = areaRetangulo(vbase, valtura);
15    vareaTri = areaTriangulo(vbase, valtura);
16
17    printf("A area do retangulo e: %f \n", vareaRet);
18    printf("A area do triangulo e: %f \n", vareaTri);
19 }
20
21 float areaRetangulo(float base, float altura) {
22     float area = base * altura;
23     return area;
24 }
25
26 float areaTriangulo(float base, float altura) {
27     float area = areaRetangulo(base, altura) / 2;
28     return area;
29 }
```

A única diferença para o código do exemplo anterior é a linha 27, note que a multiplicação da **base** por **altura** foi substituída pela invocação da função **areaRetangulo** e após esta função retornar o resultado da multiplicação, este será dividido por 2 para depois ser atribuído o resultado final à variável **area**.

## 7.5 Escopo das Variáveis

Entenda como sendo o escopo de uma variável, a área em que se poderá atribuir ou obter valores de uma determinada variável. Ou seja, a área em que a variável terá validade e significado. Basicamente, há dois tipos de escopo que uma variável pode abranger, **local** ou **global**. As **variáveis globais** podem ser acessadas de qualquer ponto do código-fonte, ou seja, ela tem validade em qualquer função presente no código. Para que uma variável seja global, é necessário declarar a mesma fora das funções presentes no código. Ao alterar o valor de uma variável global, este valor será automaticamente válido em qualquer parte do código, desta forma, se duas ou três funções utilizam uma mesma variável global, então, ao obter o valor da variável, todas funções irão receber o mesmo valor (LAUREANO, 2005).

As **variáveis locais** tem validade apenas na função em que foi declarada, ou seja, pode-se obter ou atribuir valores, apenas na função em que foram declaradas, assim, se o código-fonte possui, por exemplo, duas funções com variáveis locais com o mesmo nome, cada uma poderá assumir valores distintos em seu escopo (LAUREANO, 2005). Até o momento, todos os códigos apresentados fizeram uso de variáveis locais. Veja a seguir um exemplo de uso de variáveis **locais** e **globais**.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 float vbase, valtura;
5
6 float areaRetangulo();
7 float areaTriangulo();
8
9 void main()
10 {
11     float vareaRet, vareaTri;
12     printf("Informe a base do retangulo: \n");
13     scanf("%f", &vbase);
14     printf("Informe a altura do retangulo: \n");
15     scanf("%f", &valtura);
16     vareaRet = areaRetangulo();
17     vareaTri = areaTriangulo();
18
19     printf("A area do retangulo e: %f \n", vareaRet);
20     printf("A area do triangulo e: %f \n", vareaTri);
21 }
22
23 float areaRetangulo() {
24     float area = vbase * valtura;
25     return area;
26 }
27
28 float areaTriangulo() {
29     float area = areaRetangulo() / 2;
30     return area;
31 }
```

No código apresentado como exemplo, foram realizados vários ajustes para mostrar o uso prático das variáveis globais e locais. As variáveis **vbase** e **valtura**, que antes haviam sido declaradas na linha 11, foram declaradas na linha 4, para que as mesmas

tenham escopo de global, note que, ambas estão fora das funções **main**, **areaRetangulo()** e **areaTriangulo()**, assim elas terão validade em todas essas funções. Note que nas linhas 6 e 7, o protótipo das funções **areaRetangulo()** e **areaTriangulo()** não têm mais os argumentos, pois como as variáveis **vbase** e **valtura** são globais, então os seus valores podem ser acessados em qualquer ponto, não sendo necessário a passagem do valor por referência como antes. Veja nas linhas 13, 15 e 24 como essas variáveis são acessadas, basta utiliza-las nas funções normalmente. Agora observe que nas linhas 16, 17 e 29 as funções são invocadas sem a passagem dos parâmetros, pois como já foi dito, os valores das variáveis **base** e **altura** são agora acessadas diretamente na função e não por referência como antes.

Por outro lado, algumas variáveis permaneceram como locais, como é o caso da variável **area** que é declarada nas duas funções, **areaRetangulo()** e **areaTriangulo()**, assim, o escopo dessas variáveis é local na função em que foram declaradas, e embora tenham o mesmo nome, ambas podem assumir valores diferentes, cada uma em seu escopo.

## 7.6 Resumo da Aula

Na aula 7 foram apresentados conceitos que permitem utilizar as funções já disponíveis na linguagem C e conceitos que permitem definir nossas próprias funções. Inicialmente foram apresentadas as funções para cálculo de raiz quadrada, **sqrt()**, e cálculo de potência, **pow()**. Foram apresentadas também as funções para tratamento de texto, **strcpy()** que possibilita atribuir os valores de um vetor de **char** a outro vetor de **char** de dimensão compatível. E foi também apresentada a função **strcmp()** que permite comparar os conteúdos de dois vetores de **char**. É importante que antes de iniciar a implementação de uma nova função, fazer sempre uma pesquisa para identificar se o problema que deseja-se resolver, já não tenha uma solução em uma função disponível.

Sobre a definição de funções, primeiro foi discutida a forma geral de uma função, aspectos como o tipo de retorno da função, o nome da função, a lista de argumentos de uma função e o corpo da função, assim, para cada uma dessas características foram discutidas as regras gerais para a definição e o formato de uso. Foram apresentados alguns exemplos de uso de funções para esclarecer variados aspectos como, o uso da função, o uso de protótipos e a declaração de variáveis locais e globais.

## 7.7 Exercícios da Aula

Os exercícios desta lista foram Adaptados de [Lopes e Garcia \(2002, p. 42-61; 400-438\)](#).

1. Faça um programa em C que leia três números e, para cada um, imprimir o dobro. O cálculo deverá ser realizado por uma função e o resultado impresso ao final do programa.
2. Faça um programa que receba as notas de três provas e calcule a média. Para o cálculo, escreva uma função. O programa deve imprimir a média ao final.
3. Faça um programa em C que leia o valor de um ângulo em graus e o converta, utilizando uma função, para radianos e ao final imprima o resultado. Veja a fórmula de cálculo a seguir.

$$rad = \frac{ang \times pi}{180} \quad (7.1)$$

Em que:

- rad = ângulo em radianos
  - ang = ângulo em graus
  - pi = número do pi
4. Faça um programa que calcule e imprima o fatorial de um número, usando uma função que receba um valor e retorne o fatorial desse valor.
  5. Faça um programa que verifique se um número é primo por meio de um função. Ao final imprima o resultado.
  6. Faça um programa que leia o saldo e o % de reajuste de uma aplicação financeira e imprimir o novo saldo após o reajuste. O cálculo deve ser feito por uma função.
  7. Faça um programa que leia a base e a altura de um retângulo e imprima o perímetro, a área e a diagonal. Para fazer os cálculos, implemente três funções, cada uma deve realizar um cálculo específico conforme solicitado. Utilize as fórmulas a seguir.

$$perimetro = 2 \times (base + altura) \quad (7.2)$$

$$area = base \times altura \quad (7.3)$$

$$diagonal = \sqrt{base^2 + altura^2} \quad (7.4)$$

8. Faça um programa que leia o raio de um círculo e imprima o perímetro e a área. Para fazer os cálculos, implemente duas funções, cada uma deve realizar um cálculo específico conforme solicitado. Utilize as fórmulas a seguir.

$$perimetro = 2 \times pi \times raio \quad (7.5)$$

$$area = pi \times raio^2 \quad (7.6)$$

9. Faça um programa que leia o lado de um quadrado e imprima o perímetro, a área e a diagonal. Para fazer o cálculo, implemente três funções, cada uma deve realizar um cálculo específico conforme solicitado. Utilize as fórmulas a seguir.

$$perimetro = 4 \times lado \quad (7.7)$$

$$area = lado^2 \quad (7.8)$$

$$diagonal = lado \times \sqrt{2} \quad (7.9)$$

10. Faça um programa que leia os lados **a**, **b** e **c** de um paralelepípedo e imprima a diagonal. Para fazer o cálculo, implemente uma função. Utilize a fórmula a seguir.

$$diagonal = \sqrt{a^2 + b^2 + c^2} \quad (7.10)$$

11. Faça um programa que leia a diagonal maior e a diagonal menor de um losango e imprima a área. Para fazer o cálculo, implemente uma função. Utilize a fórmula a seguir.

$$area = \frac{(diagonalMaior \times diagonalMenor)}{2} \quad (7.11)$$

12. Faça um programa que leia os catetos (dois catetos) de um triângulo retângulo e imprima a hipotenusa. Para fazer o cálculo, implemente uma função. Utilize a fórmula a seguir.

$$hipotenusa = \sqrt{cateto1^2 + cateto2^2} \quad (7.12)$$

13. Em épocas de pouco dinheiro, os comerciantes estão procurando aumentar suas vendas oferecendo desconto. Faça um programa que permita entrar com o valor de um produto e o percentual de desconto e imprimir o novo valor com base no percentual informado. Para fazer o cálculo, implemente uma função.
14. Faça um programa que verifique quantas vezes um número é divisível por outro. A função deve receber dois parâmetros, o dividendo e o divisor. Ao ler o divisor, é importante verificar se ele é menor que o dividendo. Ao final imprima o resultado.
15. Construa um programa em C que leia um caractere (letra) e, por meio de uma função, retorne se este caractere é uma consoante ou uma vogal. Ao final imprima o resultado.
16. Construa um programa que leia um valor inteiro e retorne se a raiz desse número é exata ou não. Escreva uma função para fazer a validação. Ao final imprima o resultado.
17. Implemente um programa que leia uma mensagem e um caractere. Após a leitura, o programa deve, por meio de função, retirar todas as ocorrências do caractere informado na mensagem colocando \* em seu lugar. A função deve também retornar o total de caracteres retirados. Ao final, o programa deve imprimir a frase ajustada e a quantidade de caracteres substituídos.

18. Faça um programa que leia um vetor com tamanho 10 de números inteiros. Após ler, uma função deve verificar se o vetor está ordenado, de forma crescente ou decrescente, ou se não está ordenado. Imprimir essa resposta no final do programa.
19. Faça um programa que leia um vetor com tamanho 10 de números inteiros. Após ler, uma função deve criar um novo vetor com base no primeiro, mas, o novo vetor deve ser ordenado de forma crescente. O programa deve imprimir este novo vetor após a ordenação.
20. Faça um programa que leia 20 de números inteiros e armazene em um vetor. Após essa leitura, o programa deve ler um novo número inteiro para ser buscado no vetor. Uma função deve verificar se o número lido por último está no vetor e retornar a posição do número no vetor, caso esteja, ou -1, caso não esteja.

# Recursividade

## Metas da Aula

1. Entender e praticar os conceitos do uso de recursividade na linguagem C.
2. Aplicar variadas situações relacionadas ao uso de recursividade em programação.
3. Aprender a escrever novas funções recursivas em linguagem C.

## Ao término desta aula, você será capaz de:

1. Entender os princípios do uso de funções recursivas.
2. Definir uma função recursiva.

## 8.1 Recursividade

A aula 7 apresentou os conceitos necessários para construir uma função. Uma função sempre deverá ser invocada por um método externo a ela, assim, uma função em que todas as suas chamadas são externas, são **não** recursivas ou iterativas. Um procedimento recursivo é aquele que possui em seu corpo uma ou mais chamadas a si mesmo (SZWARCFITER; MARKENZON, 2015). Ou seja, uma função recursiva possui chamadas internas além da externa.

A recursividade em um procedimento é inerente à sua natureza (EDELWEISS; LIVI, 2014, p. 418), contudo, um procedimento recursivo, em geral, pode ser implementado de forma não recursiva, ou seja, iterativa. Porque implementar um procedimento de maneira recursiva? Essa pergunta não tem uma resposta fácil, a recursividade pode apresentar vantagens, como procedimentos com código-fonte mais conciso, claro e limpo, entretanto, boa parte das vezes um procedimento iterativo equivalente pode ser mais eficiente (FEOFILOFF, 2008; SZWARCFITER; MARKENZON, 2015).

O exemplo mais simples para uma função recursiva é o cálculo do fatorial de um número  $n > 0$ . Esse cálculo é recursivo por sua natureza que envolve a multiplicação de todos os números de 1 até  $n$ . Assim, o fatorial de  $n$  é definido como o produto de todos os inteiros entre  $n$  e 1. Exemplo, o fatorial de 4 é:  $4 \times 3 \times 2 \times 1 = 24$ , o fatorial de 3 é:  $3 \times 2 \times 1 = 6$  (TENENBAUM; LANGSAM; AUGENSTEIN, 1995, p. 133). Portanto, pode-se escrever a definição desta função assim:

$$n! = 1 \text{ se } n = 0$$

$$n! = n \times (n - 1) \times (n - 2) \times \dots \times 1 \text{ se } n > 0$$

A seguir, foi disponibilizado o código para uma primeira análise em sua versão iterativa, para então avaliar a versão recursiva. Veja a seguir o código-fonte da função iterativa.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int fatorial(int n) {
5     int i, fat = 1;
6     for (i=n; i>=1; i--) {
7         fat *= i;
8     }
9     return fat;
10 }
11
12 void main()
13 {
14     int num, resul;
15     printf("Informe um numero: \n");
16     scanf("%d", &num);
17
18     resul = fatorial(num);
19
20     printf("Fatorial de %d e %d \n", num, resul);
21 }
```

A função responsável pelo cálculo do fatorial está entre as linhas 4 e 10 do código-fonte. Note que para fazer o cálculo do fatorial, foi necessário um laço **for** que percorre

todos os números de **n** até 1. O cálculo é então realizado na linha 7, durante as **n** iterações. Entre as linhas 12 e 16, temos as instruções responsáveis pela leitura de um número **n** qualquer e a invocação da função na linha 18, esta é a chamada externa que qualquer função possui, recursiva ou não. Veja que o resultado retornado pela função na linha 18 é atribuído à variável **resul** e na linha 20 o resultado é impresso. Agora analise a seguir a versão recursiva para o mesmo programa.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int fatorial(int n) {
5     if (n <= 1)
6         return 1;
7     else
8         return n * fatorial(n-1);
9 }
10
11 void main()
12 {
13     int num, resul;
14     printf("Informe um numero: \n");
15     scanf("%d", &num);
16
17     resul = fatorial(num);
18
19     printf("Fatorial de %d e %d \n", num, resul);
20 }
```

A função recursiva para o cálculo do fatorial está compreendida entre as linhas 4 e 9. Notou que não há mais laço **for**? Isso ocorre porque no caso da função recursiva, não há iterações e sim recursões, que são responsáveis pelas chamadas internas da função, isso obriga a toda função recursiva possuir uma condição de saída da função, pois caso contrário as chamadas internas seriam infinitas. Essa condição pode ser vista na linha 5 do código apresentado, note que, caso o resultado da condição  $n \leq 1$  seja verdadeiro, a linha 6 será executada, veja que nesta linha não há uma chamada interna para a função, ou seja, essa é a instrução de saída da função. Da mesma forma, se o resultado da expressão  $n \leq 1$  resultar em falso, então a linha 8 será executada, neste caso, a função é invocada novamente.

Agora, observe a linha 8 do código, notou alguma semelhança? Veja que o código é igual à definição feita para a função, isso ocorre porque, em geral, a implementação de uma função recursiva tende a ser mais intuitiva em relação ao problema. Neste caso, um fator importante nesta invocação interna, é que o **n** é decrementado, **n-1**, caso não fosse, além do cálculo errado, teríamos outra situação de *loop* infinito, pois a condição  $n \leq 1$  nunca seria verdadeira (TENENBAUM; LANGSAM; AUGENSTEIN, 1995, p. 135). O restante do programa é igual ao do exemplo anterior.

A figura 23 mostra um teste de mesa com a função fatorial recursiva, no lado esquerdo da figura 23 é apresentado o código-fonte à cada chamada e do lado direito os dados do teste de mesa relativo à cada linha do código-fonte, além disso, o teste é realizado com  $n = 3$  na invocação externa à função.

Faça uma análise cuidadosa do teste de mesa apresentado na figura 23 para entender o funcionamento de uma função recursiva. Veja que na primeira execução da função,  $n = 3$ , a condição na linha 5 retorna **falso** e portanto, a linha 6 não é executada, levando à execução das linhas 7 e 8. Observe que na linha 8, a passagem do argumento **n** para a

função fatorial é  $n - 1$ , como  $n = 3$ , então  $n = 3 - 1$ , portanto,  $n = 2$ . Ou seja, na primeira invocação interna da função, o valor de **n** foi decrementado em 1 unidade. Note que a invocação interna da função, implica em que, a função que acaba de ser invocada, deve primeiro executar para que, somente depois disso, a primeira invocação (externa) da função seja concluída (TENENBAUM; LANGSAM; AUGENSTEIN, 1995, p. 134), assim, somente depois da função executar por completo, é que ela será capaz de devolver um valor para a função que a invocou e desta forma, a função que a invocou poderá então, multiplicar esse retorno por **n**, conforme o código-fonte da linha 8.

Agora veja a invocação interna em que  $n = 2$ , novamente o retorno da linha 5 é **falso** e a linha 6 não é executada, sendo então executadas as linhas 7 e 8. Na linha 8, ocorre um novo decremento do valor de **n**, como no escopo desta invocação da função,  $n = 2$ , então, a invocação interna à função fatorial recebe  $n = 1$ . Assim, no terceiro escopo da função fatorial, em que  $n = 1$ , tem-se que o retorno do código-fonte na linha 5 é **verdadeiro** e portanto a linha 6 é executada, como no código da linha 6 não há invocação interna da função fatorial, então as iterações irão finalizar neste momento. Agora observe que, neste momento é que a última invocação da função retorna para a sua antecessora o valor **1**, que por sua vez, o multiplica por **2** e retorna à primeira invocação (a externa) o valor **2**, que o multiplica por **3**, retornando então o valor **6** para o método que fez a invocação externa à função.

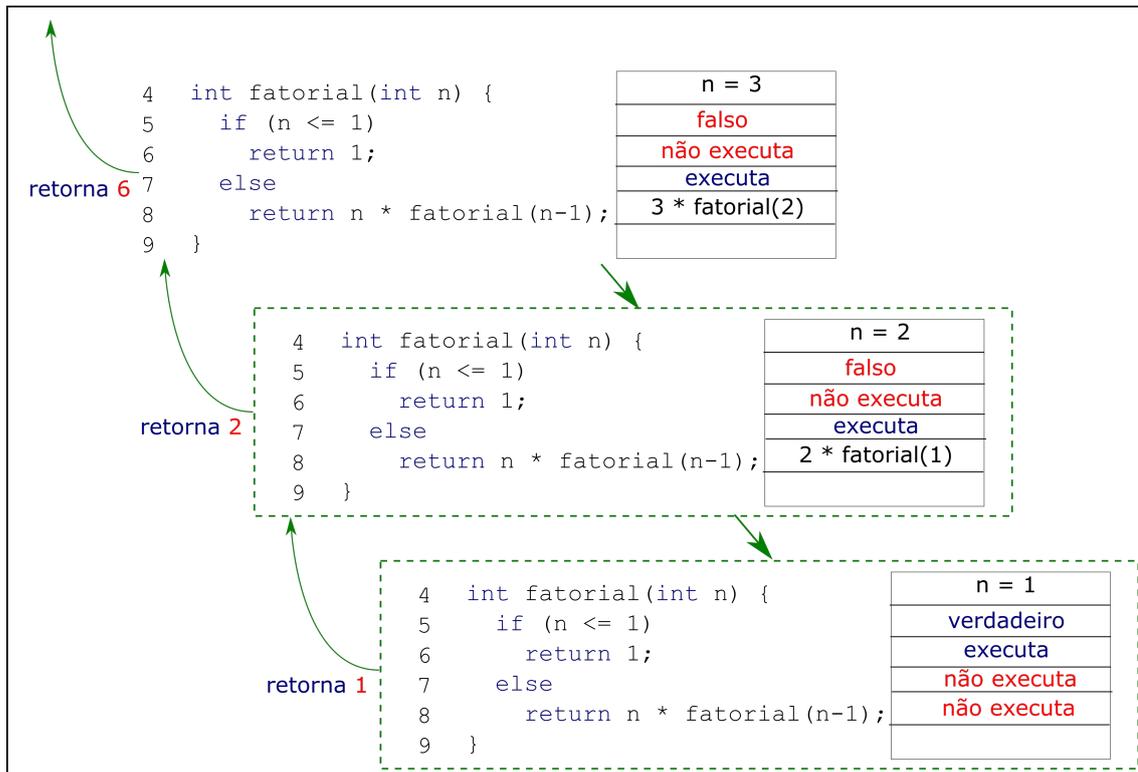


Figura 23 – Teste de mesa com a função fatorial recursiva

## 8.2 Loop infinito na recursividade

É de suma importância que um algoritmo recursivo não gere uma sequência infinita de chamadas a si mesmo (TENENBAUM; LANGSAM; AUGENSTEIN, 1995, p. 143), pois caso

ocorra o programa não irá concluir a operação e ainda, não irá terminar naturalmente, provavelmente o término será por uma interrupção forçada ou uma ou várias exceções geradas em razão de falta de memória e outros. Veja a seguir uma adaptação no código da função recursiva, de forma que, a mesma não tenha uma saída da sequência de execuções, provocando assim uma sequência infinita.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int fatorial(int n) {
5     return n * fatorial(n-1);
6 }

```

Note que agora, a função fatorial se resume ao cálculo, sem a condição para verificar se é o momento de parada, assim serão produzidas infinitas recursões até uma interrupção forçada, ou por falta de recursos do computador. A figura 24 mostra o teste de mesa para esta função.

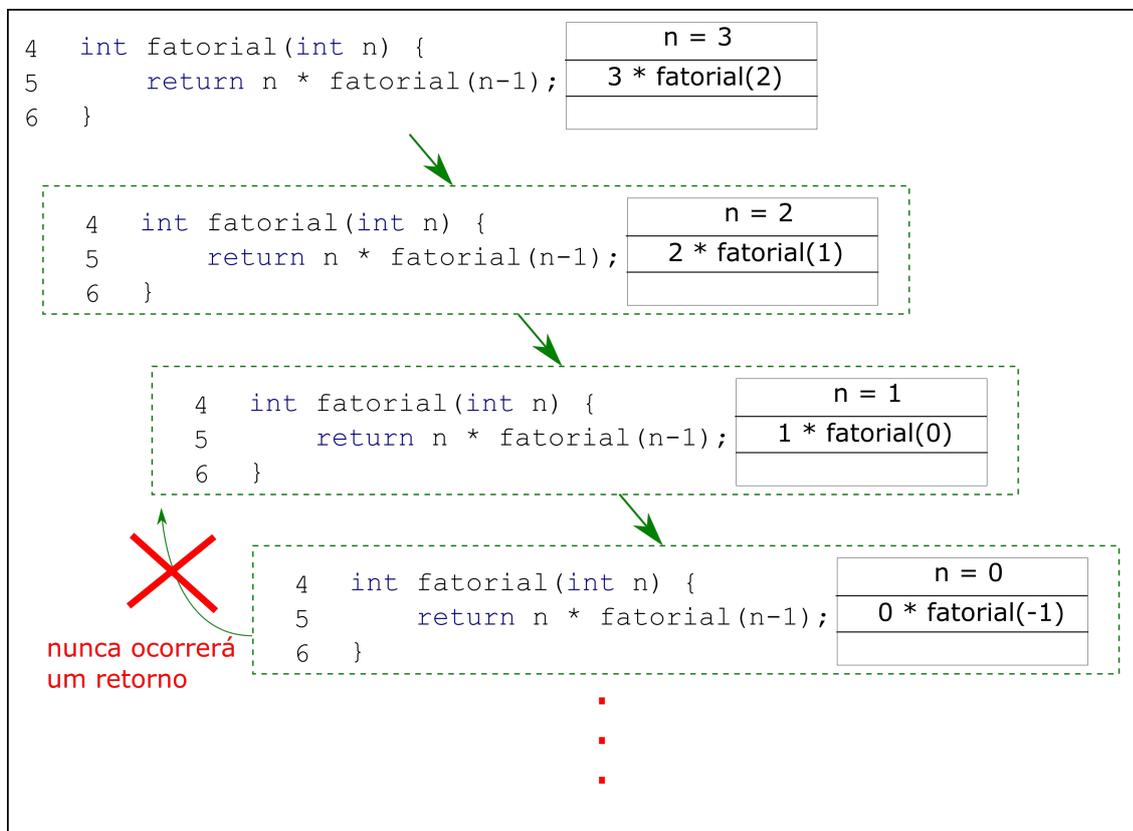


Figura 24 – Teste de mesa com a função fatorial recursiva sem interrupção das recursões

Observe na figura 24 a terceira recursão, em que o  $n = 1$ , apesar de ser desnecessário e não desejável, as recursões continuam ocorrendo, pois não há uma condição que permita a interrupção das chamadas internas à função. Assim, as chamadas contínuas provocam dois efeitos, o cálculo nunca será realizado, pois não há um retorno das funções invocadas internamente, e a execução infinita do programa, até que uma parada seja forçada ou faltem recursos ao computador para continuar executando o

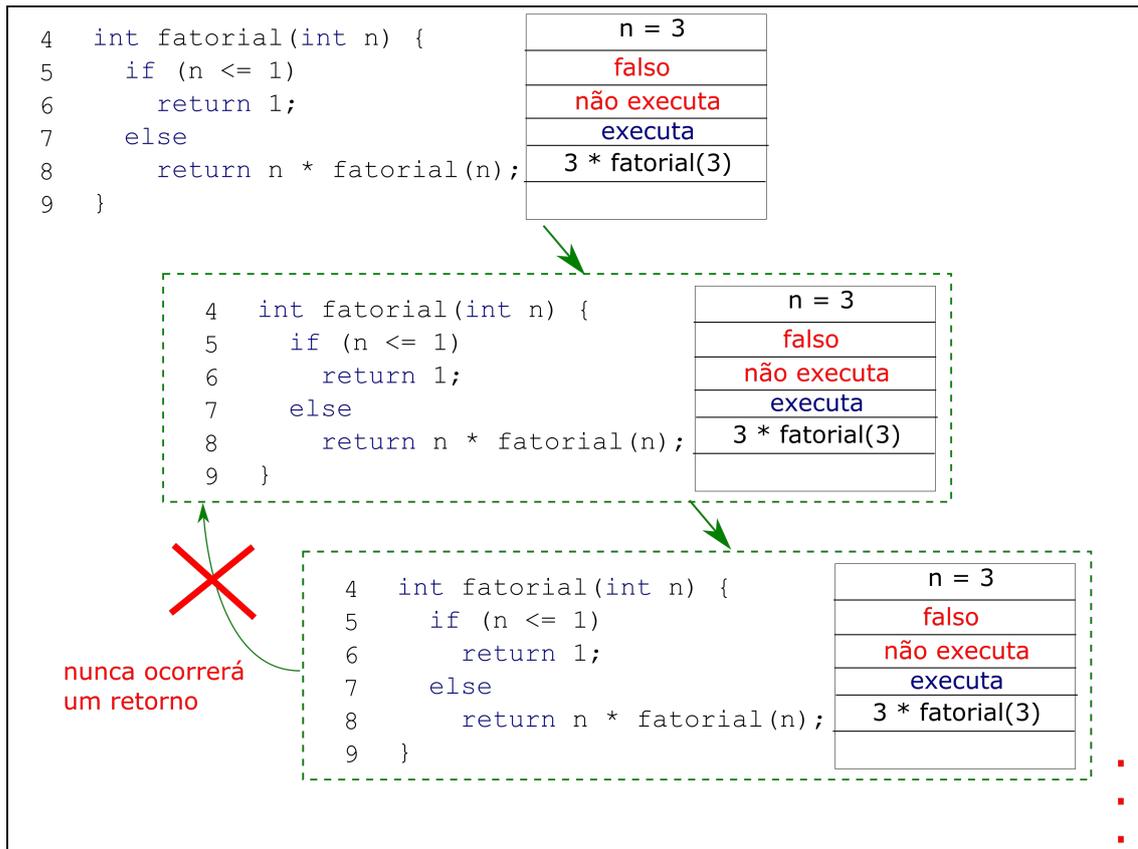


Figura 25 – Teste de mesa com a função fatorial recursiva sem interrupção das recursões

programa. Agora veja outro exemplo, na figura 25, em que uma pequena alteração pode provocar recursões infinitas.

Conforme pode ser visto na figura 25, apenas alterar o trecho **fatorial(n-1)**, para **fatorial(n)**, leva à recursões infinitas, pois neste caso, o **n** permanece inalterado nas chamadas internas da função, assim, a linha 5 do código-fonte nunca será verdadeira, o que é necessário para interromper as recursões. Similar ao exemplo anterior, neste caso também, não há retorno algum, pois nenhuma das chamadas internas obtém sucesso na conclusão e conseqüente retorno.

## 8.3 Resumo da Aula

Na aula 8 foram apresentados conceitos que permitem utilizar a recursividade em linguagem C. Inicialmente foram discutidos alguns conceitos sobre recursividade, como funções não recursivas possuem iterações enquanto funções recursivas são executadas por recursões, posteriormente, foi utilizado o exemplo recursivo do cálculo do fatorial de um número **n**.

Algumas variações do exemplo foram apresentadas para tratar do assunto *loop* infinito. Assim, foi destacado que para implementar uma função recursiva é importante que a mesma sempre tenha uma condição de saída da recursividade para evitar o *loop* infinito.

## 8.4 Exercícios da Aula

Parte dos exercícios desta lista foram Adaptados de [Tenenbaum, Langsam e Augenstein \(1995, p. 143-178\)](#) e [Edelweiss e Livi \(2014, p. 433-436\)](#).

1. Faça um programa em C que calcule, por meio de uma função recursiva,  $a \times b$  usando a adição, em que **a** e **b** são inteiros não-negativos.
2. Crie uma função recursiva que receba um número inteiro positivo **N** e calcule o somatório dos números de 1 a **N**.
3. Considere um vetor **vet** de tamanho 20. Construa um programa com algoritmos recursivos para calcular:
  - o elemento máximo do vetor;
  - o elemento mínimo do vetor;
  - a soma dos elementos do vetor;
  - o produto dos elementos do vetor;
  - a média dos elementos do vetor.
4. A sequência de Fibonacci é a sequência de inteiros: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, .... Implemente uma função recursiva que calcule e imprima todos os elementos da série Fibonacci de 0 até **n**. Em que, **n** deve ser informado pelo usuário do programa.
5. Escreva uma função recursiva em C para calcular o máximo divisor comum de dois números,  $\text{mdc}(x, y)$ .
6. Escreva um programa recursivo em linguagem C para converter um número da sua forma decimal para a forma binária. Dica: dividir o número sucessivamente por 2, sendo que o resto da *i*-ésima divisão vai ser o dígito *i* do número binário (da direita para a esquerda).
7. Escreva uma função recursiva em linguagem C para calcular o valor de  $x^n$
8. Escreva um programa em C recursivo que inverta a ordem dos elementos, números inteiros, de uma lista armazenada em um vetor. Ao final da execução, o conteúdo do primeiro elemento deverá estar no último, o do segundo no penúltimo, e assim por diante. Dica: troque os conteúdos das duas extremidades do vetor e chame uma função recursivamente para fazer o mesmo no subvetor interno.
9. Escreva uma função recursiva para calcular a função de Ackermann **A(m,n)**, sendo **m** e **n** valores inteiros não negativos, dada por:

$$\begin{aligned}
 & n + 1 && \text{se } m = 0 \\
 A(m, n) &= A(m - 1, 1) && \text{se } m > 0 \text{ e } n = 0 \\
 & A(m - 1, A(m, n - 1)) && \text{se } m > 0 \text{ e } n > 0
 \end{aligned}$$

10. Imagine que  $\text{comm}(n,k)$  representa o número de diferentes comitês de **k** pessoas, que podem ser formados, dadas **n** pessoas a partir das quais escolher. Por exemplo,  $\text{comm}(4,3) = 4$ , porque dadas quatro pessoas, **A**, **B**, **C** e **D** existem quatro possíveis comitês de três pessoas: **ABC**, **ABD**, **ACD** e **BCD**. Escreva e teste um programa

recursivo em  $\mathbb{C}$  para calcular  $\text{comm}(n,k)$  para  $n, k \geq 1$ . Para tal, considere a seguinte identidade:

$$\text{comm}(n,k) = n \quad \text{se } k = 1$$

$$\text{comm}(n,k) = 1 \quad \text{se } k = n$$

$$\text{comm}(n,k) = \text{comm}(n-1, k) + \text{comm}(n-1, k-1) \quad \text{se } 1 < k < n$$



# AULA 9

## Ponteiros

### Metas da Aula

1. Entender e praticar os conceitos do uso de ponteiros na linguagem C.
2. Aplicar variadas situações relacionadas ao uso de ponteiros em programação.
3. Aprender a escrever programas em linguagem C com uso de ponteiros.

### Ao término desta aula, você será capaz de:

1. Aplicar os princípios do uso de ponteiros.
2. Aplicar o uso de ponteiros com vetores e matrizes.
3. Utilizar ponteiros genéricos.
4. Escrever programas que utilizam ponteiros.

## 9.1 Problema

Na aula 7 falamos sobre o escopo das variáveis. Uma determinada variável só é válida no escopo em que ela foi criada, assim, uma variável declarada, por exemplo, na função **main** só poderá ser acessada na função **main**. Veja o código de exemplo a seguir, note que as variáveis **multiplicando** e **multiplicador** não podem ser acessadas na função **multiplica**, pois elas foram declaradas na função **main**.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int multiplica() {
5     //aqui nao e possivel acessar as variaveis
6     //multiplicando e multiplicador
7     int i, resultado = 0;
8     for (i=1; i<=multiplicador; i++) { //Erro ao ler essa variavel
9         resultado += multiplicando; //Erro ao ler essa variavel
10    }
11    return resultado;
12 }
13
14 void main()
15 {
16     int multiplicando, multiplicador, resultado;
17
18     printf("Informe o multiplicando: \n");
19     scanf("%d", &multiplicando);
20     printf("Informe o multiplicador: \n");
21     scanf("%d", &multiplicador);
22
23     resultado = multiplica();
24     printf("Resultado: %d \n", resultado);
25 }
```

Existem duas formas de resolver isso, a primeira foi apresentada na aula 7, trata-se da passagem de parâmetros (argumentos) para a função. Para fazer isso é só determinar na função quais são os argumentos que ela deve receber como entrada de dados e ao invocar a função, as variáveis que estão fora do escopo dela, devem ser informadas como argumentos desta função. Veja o exemplo a seguir:

O exemplo foi ajustado de forma que, agora a função **multiplica** possui dois argumentos de entrada, **multiplicando** e **multiplicador**, e ao invocar a função, na linha 21, note que as variáveis **multiplicando** e **multiplicador** declaradas na função **main** foram informadas como argumentos da função **resultado**, assim o dado armazenado nestas variáveis será copiado para os argumentos da função **multiplica**.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int multiplica(int multiplicando, int multiplicador) {
5     int i, resultado = 0;
6     for (i=1; i<=multiplicador; i++) {
7         resultado += multiplicando;
8     }
9     return resultado;
10 }
```

```
11
12 void main()
13 {
14     int multiplicando, multiplicador, resultado;
15
16     printf("Informe o multiplicando: \n");
17     scanf("%d", &multiplicando);
18     printf("Informe o multiplicador: \n");
19     scanf("%d", &multiplicador);
20
21     resultado = multiplica(multiplicando, multiplicador);
22     printf("Resultado: %d \n", resultado);
23 }
```

Uma observação importante que deve ser registrada, é que embora no exemplo os nomes utilizados para as variáveis e os parâmetros da função sejam os mesmos, no caso, **multiplicando** e **multiplicador**, não é isso que garantiu o funcionamento desta operação, ou seja, poderia, sem problemas, serem definidos nomes distintos para as variáveis e argumentos, pois como mencionado na passagem de parâmetros é realizada uma "**cópia**" do valor, assim, o valor da variável na função **main**, poderia por exemplo, ser alterado sem afetar o valor do parâmetro na função **multiplica**, ou o contrário, alterar o valor das variáveis **multiplicador** e **multiplicando** dentro da função **multiplica** e os valores das variáveis na função **main** permanecerem inalterados, pois as variáveis em questão são diferentes dos parâmetros, embora tenham os mesmos nomes.

Isso é uma coisa boa, pois este isolamento entre as funções permite uma manipulação mais eficiente das variáveis. Pois se as funções pudessem mudar os valores de uma variável seria mais difícil controlar e manipular as variáveis nos programas, tornando este trabalho mais complexo. É por isso que este tipo de passagem recebe o nome de **passagem por cópia** (ANICHE, 2015, p. 107).

A passagem por cópia é suficiente na maioria dos casos, mas em algumas situações precisamos alterar o valor da variável fora do escopo dela. Por exemplo, suponha que em um sistema financeiro seja necessário calcular os juros e atualizar a variável de saldo atual com os juros calculados, a variável com o saldo atual está no escopo da função **main** e o valor calculado de juros é retornado pela função responsável pelo cálculo de juros, então neste caso, a passagem por cópia não seria suficiente, pois a atualização do valor da variável com o saldo atual deveria ser realizada dentro da função que calcula os juros, assim, deve-se utilizar a técnica de **passagem por referência**.

Outro exemplo, suponha que você precisa executar duas operações matemáticas básicas, soma e subtração, com dois valores, como uma função não é capaz de retornar mais de um valor, então será necessário atualizar as variáveis de resultado pelos argumentos de entrada da função, novamente tem-se um caso em que é necessário utilizar a **passagem por referência**. Contudo, em C não existe passagem de parâmetros por referência (EDELWEISS; LIVI, 2014, p. 258), assim, para suprir esta necessidade utiliza-se ponteiros.

## 9.2 Ponteiros

O ponteiro é uma variável que, em vez de guardar o dado, guarda o endereço de memória, contudo, esse endereço de memória é capaz de armazenar o dado, e portanto, ao declarar um ponteiro, requer a definição de um tipo de dado, como os que já foram

apresentados na aula 1, como o **int**, **float** ou **char**. Assim, pode-se resumir que a diferença do ponteiro para a variável tradicional, é que o ponteiro é capaz de armazenar apenas o endereço de memória que será responsável por gravar o valor daquele determinado tipo (FEOFILOFF, 2008). Assim, a variável ponteiro irá "**apontar**" para aquele endereço de memória que armazena um determinado tipo de valor, exemplo, o tipo **int**. A grande dificuldade relacionada com os ponteiros é saber quando está sendo acessado o seu valor, ou seja, o endereço de memória, ou a informação apontado pelo ponteiro (LAUREANO, 2005). Essa capacidade de armazenar o endereço de memória associado à um valor, permite resolver as demandas de passagem por referência. Veja a seguir, em forma de exercício, o primeiro exemplo citado, em que deseja-se atualizar o saldo referente à um cálculo de juros realizado em uma aplicação financeira.

### 9.2.1 Exercício de Exemplo

Faça um programa em C que calcule os juros de um determinado saldo com base em uma taxa, ambos informados pelo usuário. O programa deve utilizar uma função para calcular os juros e atualizar o saldo atual com base no saldo antigo acrescido dos juros. Ao final, a aplicação deve imprimir o total de juros e o novo saldo.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 float calculaJuros(float *saldo, float taxa) {
5     float juros;
6     juros = *saldo * taxa / 100;
7     *saldo += juros;
8     return juros;
9 }
10
11 void main()
12 {
13     float vSaldo, vTaxa, vResul;
14     printf("Informe um saldo: \n");
15     scanf("%f", &vSaldo);
16     printf("Informe uma taxa: \n");
17     scanf("%f", &vTaxa);
18
19     vResul = calculaJuros(&vSaldo, vTaxa);
20
21     printf("Juros.....: %f \n", vResul);
22     printf("Saldo Novo: %f \n", vSaldo);
23 }
```

Este exemplo traz uma situação que envolve o uso das operações básicas com um ponteiro. Note que na linha 4, ao declarar os argumentos da função **calculaJuros**, foi indicado um **\*** antes da palavra **saldo**, ao fazer isso, **saldo** assume o papel de **ponteiro** para um espaço de memória que armazena valores do tipo **float**. Assim, ao invocar a função na linha 19, deve-se informar um endereço de memória do tipo **float**, conforme pode ser visto, isso foi realizado informando o símbolo **&** antes da palavra **vSaldo**. Fazendo isso, o argumento receberá o endereço de memória associado à variável **vSaldo**. Assim, quando algum acesso for realizado no argumento **\*saldo**, será por referência, realizado no endereço de memória associado à variável **vSaldo**.

Note agora que, na linha 7 o valor de **\*saldo** é atualizado, ao realizar esta operação, como **\*saldo** aponta para um endereço de memória, então o valor nesta região da

memória é atualizado. Considerando que **vSaldo**, que pertence ao escopo da função **main**, aponta para o mesmo endereço de memória, então o valor desta variável também é atualizado. Neste momento, atingiu-se o objetivo de atualizar o valor da variável **vSaldo**, fora do escopo de **main**.

## 9.3 Operações com Ponteiros

As operações básicas com ponteiros estão associadas à atribuição de valores, e como já mencionado, em se tratando de ponteiros, podemos acessar o valor armazenado por um ponteiro, ou seja, um endereço de memória, ou acessar o valor armazenado no endereço de memória no qual o ponteiro aponta. Mas, a primeira operação a ser utilizada é a declaração de ponteiros. Como um ponteiro aponta para um endereço de memória, em geral, associa-se um tipo de dado ao ponteiro, da mesma forma que associamos um tipo de dado à uma variável. Os tipos básicos de dados são apresentados na tabela 1. Veja a seguir a sintaxe para declarar um ponteiro:

```
1 //Sintaxe:
2 tipoPonteiro * nomePonteiro;
```

Como pode ser visto na sintaxe, a única diferença da declaração de variável para ponteiro, é a presença do operador **\*** antes do nome do ponteiro. A mesma regra vale para a declaração de vetores e matrizes. A operação de atribuição de valores à um ponteiro também é simples, quando a intenção é atribuir um endereço de memória, em outras palavras, apontar o ponteiro para um endereço de memória, basta fazer a atribuição da mesma forma que fazemos com uma variável, contudo, como o objetivo é atribuir um endereço de memória, então a mudança ocorre do outro lado da igualdade, pois é preciso garantir a atribuição de um endereço de memória para o ponteiro. Veja a seguir um exemplo de atribuição de um endereço de memória a um ponteiro.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int *pNum;
7     //atribuindo um endereco de memoria
8     pNum = 6356744;
9 }
```

No exemplo apresentado, o endereço de memória é representado por um número, **6356744**, quando o endereço de memória é conhecido, basta fazer a atribuição conforme o exemplo, contudo, o natural é que o endereço de memória seja desconhecido pelo programador, neste caso, como saber qual é o endereço de memória reservado para uma variável? Isso é facilmente resolvido em linguagem C com o uso do operador **&**. Veja a seguir o exemplo.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int *pNum, num;
7     //atribuindo o endereco de memoria de uma variavel
```

```

8     pNum = &num;
9 }

```

Conforme apresentado no exemplo, basta indicar o **&** antes do nome da variável, na atribuição, e será atribuído o endereço reservado para a variável, ao invés do conteúdo da variável. A segunda situação de atribuição é quando a intenção é atribuir um valor à memória na qual o ponteiro está apontando, como proceder neste caso? A solução é simples, basta indicar o operador **\*** antes do nome do ponteiro, veja um exemplo a seguir.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int *pNum, num;
7     pNum = &num;
8     //atribuindo um valor no endereço de memória
9     //no qual pNum aponta
10    *pNum = 10;
11    printf("conteudo de num: %d \n", num);
12 }

```

Observe que na linha 7 o endereço de memória da variável **num** foi atribuído à **pNum**, isso quer dizer que a partir deste momento, **pNum** aponta para **num**. Assim, na linha 10 foi atribuído, por meio do ponteiro **pNum**, o valor **10** ao endereço de memória da variável **num**, para isso, bastou indicar o **\*** antes do nome do ponteiro. Apesar de **pNum** ter recebido o valor **10**, como este ponteiro aponta para **num**, então o conteúdo da variável **num** foi atualizado, e portanto, o resultado produzido pela instrução da linha 11, será a impressão do valor **10**.

E se a intenção é obter um valor de um ponteiro? Então vale a mesma regra para a atribuição, utiliza-se o operador **\*** antes do nome do ponteiro, caso o objetivo seja obter o valor armazenado no endereço de memória, no qual o ponteiro aponta. Se a intenção é obter o endereço de memória para o qual o ponteiro aponta, basta indicar o nome do ponteiro. Veja a seguir o exemplo.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int *pNum, num;
7     pNum = &num;
8     *pNum = 10;
9     //imprimindo o endereço para o qual pNum aponta
10    printf("conteudo de pNum: %d \n", pNum);
11    //imprimindo o conteúdo do apontamento de pNum
12    printf("conteudo do apontamento de pNum: %d \n", *pNum);
13 }

```

### 9.3.1 Resumindo

A seguir foi apresentado um código-fonte de exemplo que mostra um resumo das operações necessárias para declarar ponteiros, atribuir e obter valores. Com as operações

apresentadas neste exemplo, é possível resolver a maioria dos problemas envolvendo o uso de ponteiros em C.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int *pNum; //declaracao de ponteiro
7     int num, num2;
8     pNum = &num; //aponta pNum para num
9     *pNum = 10; //atribui valor a num por meio do apontamento
10    num2 = 20; //atribui valor a num2
11    num2 += *pNum; //atribui valor a num2 por meio do ponteiro
12
13    //obtendo valor de variaveis
14    printf("conteudo de num: %d \n", num);
15    printf("endereço de num: %d \n", &num);
16    printf("conteudo de num2: %d \n", num2);
17    printf("endereço de num2: %d \n", &num2);
18
19    //obtendo valor de ponteiros
20    printf("conteudo de pNum: %d \n", pNum);
21    printf("conteudo do apontamento de pNum: %d \n", *pNum);
22    printf("endereço de pNum: %d \n", &pNum);
23 }
```

O exemplo mostra na linha 6 como declarar um ponteiro, na linha 8 como apontar um ponteiro para uma variável, na linha 9 como atribuir valor à memória, na qual o ponteiro aponta, na 11, como obter o valor da memória na qual um ponteiro aponta, e por fim, entre as linhas 14 e 22, são vários comandos **printf** que exemplificam como obter um valor ou endereço tanto de variáveis como de ponteiros.

É importante lembrar que, embora todos os exemplos apresentados façam uso de variáveis do tipo **int**, um ponteiro pode assumir vários tipos, conforme a tabela 1 ou mesmo assumir tipos definidos pelo próprio programador<sup>1</sup>, desta forma, para aplicar o aprendizado de ponteiros em outros tipos, basta fazer a troca do tipo **int** para o mais adequado ao problema.

## 9.4 Uso de Ponteiros com Vetores / Matrizes

Vetores e matrizes podem facilmente ser utilizados em conjunto com os ponteiros. Na prática cada posição de um vetor ou matriz possui um espaço de memória reservado, assim, se considerar uma posição de um vetor como uma variável, então basta aplicar os mesmos conceitos já vistos para esta posição. Contudo, podemos apontar um ponteiro para o próprio vetor ou matriz, e não somente uma posição. Veja a seguir:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
```

<sup>1</sup> Para mais detalhes sobre como definir um tipo próprio, consulte a aula 6

```
6   int *pNum; //declaracao do ponteiro
7   int vetNum[10]; //declaracao da matriz
8   pNum = &vetNum[0]; //aponta o ponteiro para uma posicao do vetor
9   *pNum = 10; //atribui valor a posicao do vetor pelo ponteiro
10
11  pNum = vetNum; //aponta pNum para o vetor vetNum
12  pNum[1] = 20; //atribui valor a posicao do vetor pelo ponteiro
13
14  printf("Posicao 1: %d \n", vetNum[0]);
15  printf("Posicao 2: %d \n", vetNum[1]);
16 }
```

Nas linhas 8 e 9 do exemplo, pode-se verificar o apontamento do ponteiro **pNum** para uma posição específica de um vetor, no caso a primeira posição, e depois a atribuição de um valor à essa posição. Essas duas instruções foram executadas seguindo as definições já vistas, a única diferença é que, como o ponteiro aponta para uma posição do vetor, então essa posição é indicada na linha 9.

Nas linhas 11 e 12, tem-se um exemplo de apontamento para o vetor e não apenas a posição, como antes, note que, utilizou-se o mesmo ponteiro, isso é permitido porque como já dito, o ponteiro armazena apenas o endereço de memória, então no caso do vetor ou matriz, ele irá armazenar o endereço da primeira posição do vetor ou da matriz, as demais posições não precisam ser armazenadas, porque elas ficam nas posições subsequentes da memória.

Ainda na linha 11, note que para atribuir o endereço de memória associado ao vetor, não foi necessário preceder o nome do vetor com o operador **&**, isso ocorre porque no caso do vetor ou matriz, ao fazer esse tipo de atribuição, o retorno padrão já é o endereço de memória relativo à primeira posição, visto que não foi indicada nenhuma posição. Na linha 12, há outra curiosidade, pois para atribuir o valor à posição do vetor não foi indicado o operador **\*** antes do nome do ponteiro, pois ao utilizar o ponteiro indicando a posição como é feito em um vetor, foi obtido o resultado equivalente.

Para utilizar ponteiros com matrizes, basta aplicar os conceitos aqui ilustrados considerando uma matriz, conforme visto na aula 5. Como apresentado o uso de ponteiros com vetores e matrizes difere muito pouco do tradicional. O uso de laços e estruturas de decisão associado à vetores e matrizes não difere do uso combinado com ponteiros.

## 9.5 Ponteiros Genéricos

Geralmente os ponteiros apontam para um tipo específico de dados, como demonstrado nos exemplos até agora, contudo, pode ocorrer de, a princípio desconhecermos o tipo de dado para o qual deseja-se que o ponteiro aponte, neste caso, é possível declarar um ponteiro *genérico*. Esse tipo de ponteiro pode apontar para qualquer tipo de dado, inclusive pode apontar para tipos de dados definidos pelo próprio programador (BACKES, 2013, p. 211). A sintaxe para declarar um ponteiro genérico é apresentada a seguir.

```
1 //Sintaxe:
2 void * nomePonteiro;
```

O uso de ponteiros genéricos requer alguns cuidados adicionais, em todas as situações que for necessário atribuir ou obter dados por meio do ponteiro. Neste caso, sempre será necessário converter o ponteiro genérico para o tipo específico em que o

dado está sendo atribuído ou obtido. A seguir é apresentado um exemplo de atribuição do valor com a conversão do ponteiro genérico.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     void *pgenerico;
7     int numero;
8     char letra;
9     pgenerico = &numero;
10    *(int*)pgenerico = 10;
11    pgenerico = &letra;
12    *(char*)pgenerico = 'a';
13
14    printf("Conteúdo da variável numero: %d \n", numero);
15    printf("Conteúdo da variável letra: %c \n", letra);
16    printf("Conteúdo da variável por meio ponteiro: %c \n", *(char*)pgenerico);
17 }
```

Na linha 6 do exemplo, foi realizada a declaração do ponteiro genérico, cujo nome é **pgenerico**, na linha 9 o ponteiro é apontado para a variável **numero**, note que o apontamento é feito da mesma forma que um ponteiro **não genérico**. Na linha 10, é atribuído o valor **10** ao conteúdo para o qual o ponteiro **pgenerico** aponta, note que neste caso, houve a necessidade de fazer a conversão colocando entre o operador **\*** e o nome do ponteiro o seguinte código: **(int\*)**. Utilizou-se o tipo **int**, naturalmente porque a variável para o qual o ponteiro **pgenerico** aponta é do tipo inteiro. Na linha 11, o mesmo ponteiro, que até então apontava para o tipo inteiro, passa a apontar para a variável **letra**, que é do tipo **char**, neste momento, é possível ver a capacidade do ponteiro genérico. Na linha 12 é utilizada a mesma estratégia para atribuir o valor **'a'**, ao conteúdo da variável que o ponteiro genérico aponta. Na linha 16, pode-se ver um exemplo de acesso do conteúdo para o qual o ponteiro aponta, note que é realizada a mesma conversão, neste caso, como o ponteiro foi apontado por último para a variável **letra**, então a conversão é feita com o tipo **char**.

Pode-se concluir então que o ponteiro genérico é poderoso por permitir alternar o apontamento entre vários tipos, o que trás várias possibilidades ao programador. Além disso, pode-se dizer que a implementação é relativamente simples, pois há pouca mudança em relação aos ponteiros de um tipo específico, a única diferença reside no fato de que é necessário realizar a conversão do ponteiro genérico ao fazer atribuição, ou obter o valor do conteúdo para o qual o ponteiro aponta.

## 9.6 Ponteiro para Ponteiro

Resumidamente, pode-se dizer que uma variável tem um espaço alocado de memória associado à ela, e um ponteiro também tem um espaço de memória associado a ele, mas que, diferente da variável, o espaço de memória do ponteiro guarda apenas um endereço de memória, ou seja, um apontamento para outra área de memória. Considerando estes fatos, é possível que um ponteiro armazene um endereço de memória, que por sua vez, possui outro endereço de memória armazenado, isso caracteriza um apontamento de ponteiro para ponteiro (BACKES, 2013, p. 217). A linguagem C permite criar ponteiros com diferentes níveis de apontamento, em outras palavras, é possível

criar um ponteiro que aponta para outro ponteiro, que por sua vez aponta para outro ponteiro e assim sucessivamente, sem restrição de níveis de apontamento. A sintaxe para declarar um ponteiro para ponteiro é:

```
1 //Sintaxe:
2 tipoPonteiro **nomePonteiro;
```

Notou a diferença? Diferente da declaração do ponteiro para variável, para declarar um ponteiro para ponteiro deve-se adicionar um operador `*` para cada nível de apontamento, ou seja, no caso do ponteiro para ponteiro, serão dois operadores `**`, se deseja declarar ponteiro para ponteiro para ponteiro, serão `***` e assim por diante, contudo, não é muito comum utilizar três níveis de apontamento ou mais. Veja a seguir um exemplo de uso.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int x;
7     int *p, **p2;
8     x = 10; //atribuindo valor a x
9     p = &x; //apontando p para x
10    p2 = &p; //apontando p2 para p
11    printf("Conteúdo de p2: %d \n", p2);
12    printf("Conteúdo em que p2 aponta: %d \n", *p2);
13    printf("Conteúdo em que p aponta: %d \n", **p2);
14 }
```

Na linha 7 do exemplo, foram declarados os dois ponteiros, **p** e **p2** que é um ponteiro para ponteiro, na linha 9 o ponteiro **p** foi apontado para a variável **x** e na linha 10, o ponteiro **p2** foi apontado para o ponteiro **p**, note que a forma de apontar um ponteiro para uma variável ou para outro ponteiro é igual. Na linha 11 foi impresso o conteúdo do ponteiro **p2**, como sabemos que é ponteiro, então o resultado dessa impressão será o endereço de memória para o qual **p2** está apontando, no caso **p**. Na linha 12 foi impresso o conteúdo do endereço que **p2** aponta, ou seja o conteúdo de **p**, como **p** é um ponteiro, então o seu conteúdo é um endereço de memória para o qual ele aponta, no caso, a variável **x**. Por fim, na linha 13 foi impresso o conteúdo do endereço de memória em que **p** aponta, ou seja, o conteúdo de **x**. O interessante dessas impressões é que todas foram realizadas por meio do ponteiro **p2**, bastou utilizar 1 operador `*` para imprimir o conteúdo em que **p2** aponta e 2 operadores `*` para imprimir o conteúdo do apontamento do ponteiro em que **p2** aponta.

## 9.7 Resumo da Aula

Na aula 9 foram apresentados conceitos que permitem utilizar **ponteiros** em linguagem C. Em geral, os ponteiros são úteis quando é necessário atualizar o valor de uma variável fora do escopo dela, ou seja, em uma função, por exemplo. Para isso, foram apresentados exemplos de situações e de código-fonte que permitem ao leitor entender como declarar um ponteiro, atribuir ou obter valores.

Nesta aula, foi possível também entender várias facetas relacionadas ao uso de ponteiros, como apontar um ponteiro para outro ponteiro, declarar um ponteiro genérico, utilizar ponteiros com vetores e matrizes, além disso, entender melhor como o computador e o compilador tratam o acesso e o armazenamento na memória do computador, pois em vários exemplos, questões relacionadas à memória foram administradas. A próxima aula, 10, vai permitir ao leitor que os conceitos acerca dos ponteiros fiquem mais claros.

## 9.8 Exercícios da Aula

Parte dos exercícios desta lista foram Adaptados de [Backes \(2013, p. 219-220\)](#) e [Edelweiss e Livi \(2014, p. 263-268\)](#).

1. Escreva um programa que contenha duas variáveis inteiras. Compare seus endereços e exiba o maior endereço.
2. Crie um programa que leia números reais em um vetor de tamanho 10. Imprima o endereço de cada posição desse vetor.
3. Crie um programa que contenha um vetor de inteiros com tamanho 5. Utilizando apenas ponteiros, leia valores e armazene neste vetor e após isso, imprima o dobro de cada valor lido.
4. Elabore um programa que leia um valor do tipo inteiro e, por meio de função, atualize todas as posições de um vetor com o número inteiro lido, depois imprima os valores. Utilize ponteiros para as operações.
5. Faça um programa que receba dois valores inteiros, após receber esses dois valores, uma função deve calcular e retornar para o programa o resultado da soma e da subtração dos valores. Obs.: Apenas uma função deve realizar esta operação, desta forma, faça uso de ponteiros.
6. Construa uma função que, recebendo como parâmetros quatro números inteiros, devolva ao módulo que o chamou os dois maiores números dentre os quatro recebidos. Faça um programa que leia tantos conjuntos de quatro valores quantos o usuário deseje e que acione a função para cada conjunto de valores, apresentando a cada vez os dois maiores números. Se preferir, utilize vetor para armazenar o conjunto de valores.
7. Considere um vetor de 10 elementos, contendo valores inteiros. Faça um programa que leia os valores para preencher esse vetor, após a leitura o programa deve invocar uma função que calcule e devolva as frequências absoluta e relativa desses valores no conjunto. (Observação: a frequência absoluta de um valor é o número de vezes que esse valor aparece no conjunto de dados; a frequência relativa é a frequência absoluta dividida pelo número total de dados.). Utilize outros dois vetores para armazenar as frequências relativas e absolutas e ao final do programa, imprima de forma tabulada os números e suas frequências absoluta e relativa.
8. O laboratório de agropecuária da Universidade Federal do Capa Bode tem um termômetro de extrema precisão, utilizado para aferir as temperaturas de uma estufa onde cultivam uma variedade de jaca transgênica, com apenas um caroço do tamanho de uma semente de laranja. O problema é que este termômetro dá os resultados na escala Kelvin (K) e os pesquisadores que atuam perto da estufa são americanos, acostumados com a escala Fahrenheit (F). Você deve criar um programa para pegar uma lista de 24 temperaturas em Kelvin e convertê-las para Fahrenheit. O problema maior é que esses pesquisadores querem que você faça essa conversão e imprima os resultados utilizando ponteiros. Para a conversão, observe as fórmulas a seguir:

$$F = 1.8 \times (K - 273) + 32 \quad (9.1)$$

Em que:

- F = Fahrenheit
  - K = Kelvin
9. A Google está desenvolvendo um novo sistema operacional para máquinas de venda de bolinhas de borracha de R\$1,00, mas precisa realizar testes no Gerenciador de Memória desse novo sistema. Você foi contratado para fazer um programa para verificar se o gerenciador de memória está funcionando corretamente. Seu programa deverá ler 3 números inteiros, 3 números decimais, 3 letras, armazená-las em variáveis, e depois, através de ponteiros, trocar os seus valores, substituindo todos os números inteiros pelo número 2014, os decimais por 9.99, e as letras por 'Y'. Depois da substituição, o programa deverá exibir o valor das variáveis já devidamente atualizados.
  10. O departamento comercial da Batatinha S/A necessita atualizar os valores de seus produtos no seu catálogo de vendas. O presidente ordenou um reajuste de 4.78% para todos os itens. São 15 itens no catálogo. Sua tarefa é elaborar um programa que leia o valor atual dos produtos e armazene em um vetor, e após isso efetue o reajuste no valor dos produtos. O reajuste (acesso ao vetor) deverá ser feito utilizando ponteiros. Imprima na tela o valor reajustado, usando também ponteiros.



# **Alocação Dinâmica de Memória**

## Metas da Aula

1. Entender os conceitos necessários para realizar alocação dinâmica em linguagem C.
2. Aprender a definir o tamanho necessário de memória a ser reservada.
3. Aplicar variadas situações relacionadas ao uso de alocação dinâmica em programação.
4. Aprender a escrever programas em linguagem C com alocação dinâmica de memória.

## Ao término desta aula, você será capaz de:

1. Aplicar o uso de alocação dinâmica.
2. Determinar o tamanho da memória ser alocada.
3. Escrever programas que utilizam alocação dinâmica de memória.

## 10.1 Alocação Estática de Memória

Antes de iniciar o tópico sobre alocação dinâmica de memória, é importante discutir sobre a alocação estática, pois isso facilitará o entendimento de "quando" aplicar os conceitos de alocação dinâmica.

Até este ponto da leitura, todos os conceitos abordados estão relacionados à alocação estática de memória, pois quando, por exemplo, declaramos uma variável do tipo `int`, estamos reservando espaço de memória para aquela variável. O tamanho dessa memória reservada, varia de acordo com o tipo do dado e o compilador, mas em geral, segue a quantidade descrita na tabela 10 apresentada na aula 4. O tipo `int`, por exemplo, reserva um espaço de **16 bits**. Desta forma, se declarar um vetor de inteiros com tamanho 10, então o nosso vetor irá reservar  $10 \times 16 = 160$  bits.

Mas, a questão é: Como saber que os 16 bits serão suficientes para reservar o dado? E ainda, como saber se as 10 áreas de memória reservadas são suficientes? Será que não seriam necessárias 20 áreas? Ou 50? Veja um exemplo de problema. Imagine que você foi solicitado a construir um programa para o cadastro dos dados dos funcionários de uma pequena empresa em expansão, com cerca de 30 funcionários. Como a empresa tem apenas 30 funcionários, você provavelmente não vai precisar reservar mais do que uma área de memória para os dados de 50 funcionários, pelo menos por um tempo, mas, como garantir isso? De fato, não há como. A empresa pode expandir rapidamente e ter que contratar 100 novos funcionários para atender há um contrato, por exemplo. Bem, então uma opção seria reservar uma área suficiente para armazenar os dados de 1000 funcionários. Neste caso, pode ser que seja suficiente por muito tempo, contudo, pode ocorrer um outro efeito indesejado, que é o desperdício de memória, ou seja, enquanto a empresa não atingir um número considerável de funcionários, o programa irá consumir muita memória desnecessariamente, memória que não será utilizada na prática, mas que não estará disponível para outros programas, que possam vir a precisar. Além disso, se algum dia a empresa superar os 1000 funcionários, o programa não atenderá mais a empresa.

Essa então é a questão que nos leva a utilizar a alocação dinâmica de memória. Com esta técnica, podemos alocar somente a memória necessária para o momento, e a medida que essa necessidade aumenta, alocamos mais memória, assim, não haverá falta de memória, nem desperdício.

## 10.2 Alocação Dinâmica de Memória

Dado o problema apresentado, pense! Em que momento é possível saber se uma empresa tem 20, 50 ou 1000 funcionários? No momento em que o programador escreve o programa? Certamente, que não! Um programa de computador não se deteriora, desta forma, uma empresa poderá utilizar um programa por anos e anos, e neste caso, o cenário da empresa em relação à quando o programa foi construído, pode mudar muito. Assim, o momento certo para determinar a quantidade necessária de memória a ser reservada, é durante a execução do programa. Ou seja, ao executar o programa, será reservada a memória necessária e esta será aumentada ou diminuída conforme a necessidade, durante a execução do programa (EDELWEISS; LIVI, 2014, p. 396). Esse processo recebe o nome de alocação dinâmica, pois, a mudança do tamanho da memória reservada ocorre durante o ciclo de execução do programa.

A linguagem C permite alocar memória dinamicamente utilizando ponteiros (BACKES, 2013, p. 222). A alocação dinâmica permite ao programador declarar vetores e matrizes dimensionando o tamanho em tempo de execução, conforme a necessidade do programa em um determinado momento. Assim, ao invés de determinar um tama-

no fixo para o vetor, 1000 por exemplo, ele pode declarar o vetor com um tamanho **curinga**, que permitirá esse dimensionamento conforme a demanda. Nesta aula, serão apresentados os conceitos que permitirão entender como fazer isso na prática.

Antes de começar é importante entender na prática como isso ocorre em termos computacionais. É muito simples, como já mencionado, deve-se utilizar um ponteiro para fazer a alocação dinâmica, pois ao requisitar um espaço de memória ao sistema operacional em tempo de execução usando um ponteiro, ele irá devolver para o programa o endereço do início deste espaço de memória que foi alocado (BACKES, 2013; EDELWEISS; LIVI, 2014).

A figura 26 mostra um exemplo em que foi requisitada a alocação de 5 posições do tipo **int** ao sistema operacional. Veja que inicialmente o ponteiro é declarado e o seu conteúdo aponta para NULL, posteriormente o sistema operacional devolve para o ponteiro o endereço do primeiro espaço de memória reservado, assim, o ponteiro passa a se comportar como um vetor de tamanho 5, em que, em cada posição, podem ser armazenados valores, assim como em um vetor. A mesma dinâmica vale para matrizes também.

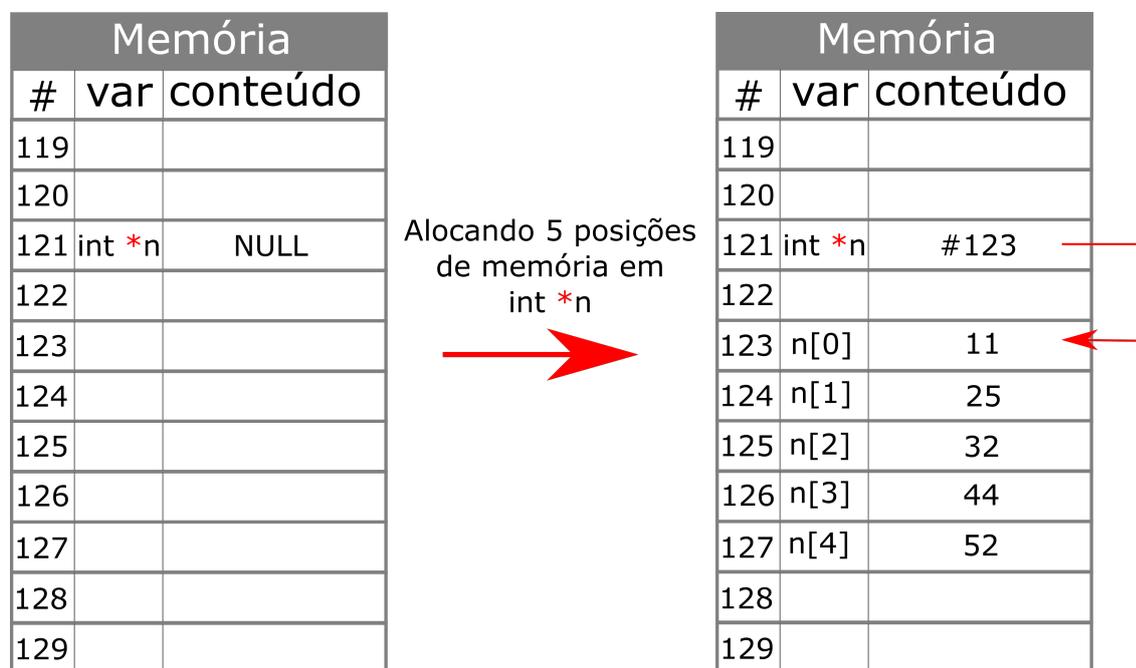


Figura 26 – Representação didática de alocação dinâmica

Fonte: Adaptado de (BACKES, 2013, p. 223)

## 10.3 Implementação de Alocação Dinâmica

Para implementar um programa com recursos de alocação dinâmica, em geral, 5 funções da biblioteca **stdlib.h** serão úteis:

- malloc
- calloc
- realloc

- free
- sizeof

Cada função tem seu papel, desta forma, é apresentado a seguir uma descrição breve sobre cada função, e após a definição de cada uma, são apresentados alguns exemplos de implementação.

### 10.3.1 Função *sizeof()*

A função **sizeof()** não lida propriamente com alocação dinâmica, mas ela é muito útil ao utilizar as demais funções, por isso, tratou-se dela primeiro. A função **sizeof()** possui a capacidade de retornar o tamanho que um tipo de dado ocupa na memória, isso é muito útil, pois como já dito o tamanho que um tipo ocupa na memória pode variar de acordo com o tipo e o compilador, a tabela 10 mostra, por exemplo, a variação de tamanho entre os tipos.

Quando alocamos um espaço de memória dinamicamente, fazemos isso para um tipo de dado primitivo, conforme a tabela 1, ou um tipo definido pelo programador<sup>1</sup>, em ambos os casos, é preciso saber qual o tamanho que este determinado tipo ocupa em memória. Como este tamanho pode variar entre compiladores da linguagem C, a função **sizeof()** torna essa questão um mero detalhe, pois ao utilizar a função para determinar o tamanho do tipo, o programa se torna independente de compiladores e arquitetura. A seguir um exemplo de uso da função.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int num, tamanho;
7     char letra;
8     tamanho = sizeof(num);
9     printf("Tamanho em bytes do inteiro: %d \n", tamanho);
10
11     tamanho = sizeof(letra);
12     printf("Tamanho em bytes do char: %d \n", tamanho);
13 }
```

Veja que, na linha 8 a função **sizeof()** foi utilizada para obter o tamanho em bytes da variável **num** do tipo **int**, conforme pode ser visto na linha 6. Após obter o tamanho, o valor é impresso na linha 9. Novamente a função é invocada na linha 11, desta vez para obter o tamanho da variável **letra** do tipo **char**. Os dois tipos foram utilizados ao invocar a função para mostrar que ela é capaz de obter o tamanho de qualquer variável de qualquer tipo. Veja também que a forma de uso da função é bem simples, basta informar, a variável que deseja obter o tamanho, como argumento e a função irá retornar o tamanho em bytes.

### 10.3.2 Função *malloc()*

A função **malloc()** é uma das funções responsáveis pela alocação de memória em tempo de execução. É a mais utilizada. O modo de funcionamento dela é: solicita ao sistema

<sup>1</sup> Para mais detalhes sobre como definir um tipo próprio, consulte a aula 6

operacional a alocação de memória e retorna um ponteiro com o endereço do início da área de memória reservada (BACKES, 2013, p. 224). A seguir a sintaxe da função **malloc()**.

```
1 //Sintaxe:
2 #include <stdlib.h>
3 void *malloc(unsigned int num);
```

Conforme pode ser visto na sintaxe, a função **malloc()** recebe um parâmetro de entrada, **num**, referente ao tamanho do espaço de memória que deve ser alocado. Além disso, a função retorna um ponteiro para a primeira posição do vetor alocado ou **NULL**, caso ocorra algum erro. Note que, o retorno da função **malloc()** é um ponteiro genérico (**void\***), pois ela não sabe que será feito com a memória alocada (BACKES, 2013, p. 225). Veja a seguir um exemplo de uso da função.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int *p;
7     p = malloc(10 * 4);
8 }
```

Na linha 6 foi declarado um ponteiro do tipo **int**, neste caso, ao alocar a memória deve-se preocupar com o tamanho em bytes necessário para o tipo inteiro. Na linha 7 foi utilizada a função **malloc()** para alocar memória para **p**, note que o argumento **num** da função, em que deve-se informar o tamanho necessário de memória para alocar, recebeu os valores **10 \* 4**, poderia ter sido informado simplesmente 40, mas o objetivo foi destacar que está sendo solicitado **10 \* 4 bytes**, sendo 4 bytes o tamanho necessário para o inteiro, então, o resultado será um vetor de inteiro com tamanho 10. Por isso, a função **sizeof()** é muito útil, pois como saber que o inteiro requer **4 bytes**? Veja a seguir o exemplo ajustado com a função **sizeof()**.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int *p;
7     p = malloc(10 * sizeof(int));
8 }
```

Note que na linha 7, o valor 4, referente aos bytes foi substituído pelo código **sizeof(int)**, este código irá retornar o tamanho correto em bytes para qualquer tipo, independente de compilador ou arquitetura, isso trará a característica de portabilidade ao código. Como já mencionado, a função **malloc()** retorna um ponteiro genérico, pois ela não sabe o que pretende-se fazer com a memória alocada, desta forma, é importante garantir que a memória alocada suportará o tipo desejado, para isso, basta fazer a conversão do retorno da função **malloc()**. Veja a seguir o exemplo ajustado com a conversão.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
```

```

4 void main()
5 {
6     int *p;
7     p = (int*) malloc(10 * sizeof(int));
8 }

```

Novamente, a mudança ocorreu na linha 7, note que agora foi adicionado o código **(int\*)** antes da função **malloc()**, esse código fará a conversão do tipo genérico para **int**, naturalmente, os exemplos com o tipo **int**, podem ser adaptados para outros tipos, apenas trocando o operador de tipo. Outro detalhe já mencionado, é que, em caso de erro ao alocar a memória, a função **malloc()** retornará **NULL**, assim, é fácil verificar, antes de tentar utilizar o ponteiro, se a memória foi alocada com sucesso. Veja a seguir um exemplo.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int *p;
7     p = (int*) malloc(10 * sizeof(int));
8     if (p == NULL) {
9         printf("Erro: Memoria insuficiente!\n");
10    }
11    else {
12        int i;
13        for (i=0; i<10; i++) {
14            printf("Informe um valor: \n");
15            scanf("%d", &p[i]);
16        }
17    }
18 }

```

Observe que da linha 8 em diante, foram adicionadas instruções para exemplificar o tratamento em caso de erro ao alocar memória. Na linha 8 foi adicionada uma condição que verifica se **p == NULL**, pois caso esta verificação seja verdadeira, significa que não houve sucesso ao alocar a memória, neste caso, será executada a linha 9 que imprime uma mensagem de erro. Se a verificação da linha 8 não retornar erro, então as linhas 12 à 16 serão executadas.

### 10.3.3 Função *calloc()*

A função **calloc()** tem o mesmo papel da função **malloc()**, ou seja, alocar memória dinamicamente. Contudo, há duas diferenças, primeiro, a função **calloc()** requer dois argumentos, o que torna explícito o tamanho requisitado para a memória e o tamanho de cada bloco da memória requisitada, segundo, a função **calloc()** inicializa os blocos de memória alocados de acordo com o tipo, se o tipo é inteiro, por exemplo, então cada bloco será inicializado com **0** (zero). Isso, naturalmente, requer mais esforço da função **calloc()**, desta forma, para grandes espaços de memória alocados, a função **malloc()** apresenta um melhor desempenho. A seguir a sintaxe da função **calloc()**.

```

1 //Sintaxe:
2 #include <stdlib.h>
3 void *calloc(unsigned int num, unsigned int size);

```

Conforme a sintaxe, a função **calloc()** recebe dois parâmetros de entrada, **num**, referente ao tamanho do espaço de memória que deve ser alocado e **size** referente ao tamanho de cada bloco da memória a ser alocada. Assim como a função **malloc()**, **calloc()** retorna um ponteiro para a primeira posição do vetor alocado ou **NULL**, caso ocorra algum erro. Veja a seguir um exemplo de uso da função.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int *p;
7     p = (int*) calloc(10, sizeof(int));
8     if (p == NULL) {
9         printf("Erro: Memoria insuficiente!\n");
10    }
11    else {
12        int i;
13        for (i=0; i<10; i++) {
14            printf("Informe um valor: \n");
15            scanf("%d", &p[i]);
16        }
17    }
18 }
```

Veja que a linha 7, do código-fonte de exemplo, é a responsável pela invocação da função **calloc()**, note que a diferença é que neste caso, são dois argumentos, ao invés de multiplicar **10** pelo tamanho do bloco de memória, como é feito com a função **malloc()**, neste caso, informa-se **10** como primeiro argumento, e o tamanho do bloco de memória como segundo argumento. Outro ponto a ser observado é que, a função também retorna um **NULL**, caso ocorra algum erro ao alocar a memória, permitindo assim verificar se houve sucesso, como pode ser visto na validação que é realizada a partir da linha 8.

### 10.3.4 Função *realloc()*

A função **realloc()** é capaz de alocar ou realocar blocos de memória já alocados pelas funções **malloc()**, **calloc()** ou a própria função **realloc()**. Mas o que é realocar? Basicamente é mudar o tamanho da memória que já foi alocada, imagine a situação em que você alocou memória, mas agora você precisa alocar mais, pois a memória anterior está acabando, então neste caso, utilize **realloc()**. Veja a seguir a sintaxe:

```
1 //Sintaxe:
2 #include <stdlib.h>
3 void *realloc(void *ptr, unsigned int num);
```

Imagine o seguinte, se a função **realloc()** tem o papel de realocar blocos de memória já alocados, então o que é primordial informar a esta função para que ela consiga desempenhar o seu papel? Isso mesmo, o ponteiro que aponta para o primeiro bloco de memória que já foi alocada, este é o primeiro argumento da função **realloc()**, o segundo argumento, é o tamanho, no qual se deseja realocar a memória. Veja a seguir um exemplo de uso da função.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int *p;
7     p = (int*) malloc(10 * sizeof(int));
8     if (p == NULL) {
9         printf("Erro: Memória insuficiente!\n");
10    }
11    else {
12        int i;
13        for (i=0; i<10; i++) {
14            printf("Informe um valor: \n");
15            scanf("%d", &p[i]);
16        }
17    }
18    //aumentando o tamanho da memória
19    p = realloc(p, 20 * sizeof(int));
20
21    if (p == NULL) {
22        printf("Erro: Memória insuficiente!\n");
23    }
24    else {
25        int i;
26        for (i=0; i<20; i++) {
27            printf("Valor na posição %d: %d\n", i, p[i]);
28        }
29    }
30 }
```

Como pode ver no código-fonte de exemplo, na linha 7 a memória foi normalmente alocada com a função **malloc()**, sendo um vetor do tipo **int** com tamanho 10, esta área é então utilizada e depois é realocada com tamanho 20 pela função **realloc()**, como pode ser visto na linha 19. Note que a função **realloc()** requer dois argumentos, o primeiro é o próprio ponteiro **p**, que já apontava para um bloco de memória reservada e o segundo argumento é o novo tamanho, no qual essa memória já reservada, deve passar a ter. Depois de realocada a memória, o ponteiro pode novamente ser utilizado normalmente como um vetor, como pode ser visto a partir da linha 21. Um detalhe importante sobre a função **realloc()** é que, ao realocar blocos de memória já previamente alocados, os dados já gravados naquele trecho não serão perdidos. Além disso, a função **realloc()** também é capaz de diminuir o tamanho da memória alocada, mas neste caso, dados podem ser perdidos.

### 10.3.5 Função *free()*

A função **free()** possui um papel muito importante, o de liberar os espaços de memória alocados dinamicamente, para uso de outros programas ou processos. Isso é necessário, pois diferente das variáveis declaradas de forma estática, a memória alocada dinamicamente não é liberada automaticamente pelo programa, mesmo após este ser encerrado. Assim, o sistema operacional não tomará conhecimento de que aquela área de memória está novamente disponível para uso. Veja a seguir a sintaxe da função **free()**.

```
1 //Sintaxe:
2 #include <stdlib.h>
3 void free(void p);
```

Como pode ser visto na sintaxe, a função `free()` é muito simples, possui um único argumento de entrada, o ponteiro no qual deseja-se que a memória seja disponibilizada para uso ao sistema operacional. Veja a seguir um exemplo de uso.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int *p;
7     p = (int*) malloc(10 * sizeof(int));
8     if (p == NULL) {
9         printf("Erro: Memoria insuficiente!\n");
10    }
11    else {
12        int i;
13        //inserindo informacoes
14        for (i=0; i<10; i++) {
15            printf("Informe um valor: \n");
16            scanf("%d", &p[i]);
17        }
18        //imprimindo informacoes
19        for (i=0; i<10; i++) {
20            printf("Valor: %d \n", p[i]);
21        }
22    }
23    //liberando a memoria alocada para p
24    free(p);
25 }
```

Note no exemplo que após utilizar a memória alocada para o ponteiro `p`, esta memória foi liberada, conforme pode ser visto na linha 24. Para isso, bastou invocar a função `free()` passando como argumento o ponteiro `p`.

## 10.4 Alocação Dinâmica de Matrizes

Até o momento os exemplos discutidos alocam um vetor dinamicamente, ou seja, um estrutura de 1 dimensão. Mas, como funciona alocação de estruturas matriciais, ou seja, estruturas com mais de uma dimensão? Neste caso, será necessário utilizar o conceito de **ponteiro para ponteiro** apresentado na aula 9, pois neste caso, como as duas dimensões (ou mais) serão dinâmicas, então será necessário construir uma estrutura como a representada na figura 27.

No lado esquerdo da figura 27 foi representada a primeira dimensão que é construída da seguinte forma, primeiro aloca-se dinamicamente um **ponteiro para ponteiro**, desta forma, na prática teremos uma vetor de ponteiros que fará o papel das linhas da matriz. Após ter esse vetor de ponteiros, pode-se alocar dinamicamente, para cada posição do vetor de ponteiros, as colunas da linha, como é representado na parte direita da figura 27. Para fazer isso, é preciso utilizar laços aninhados, no primeiro laço,

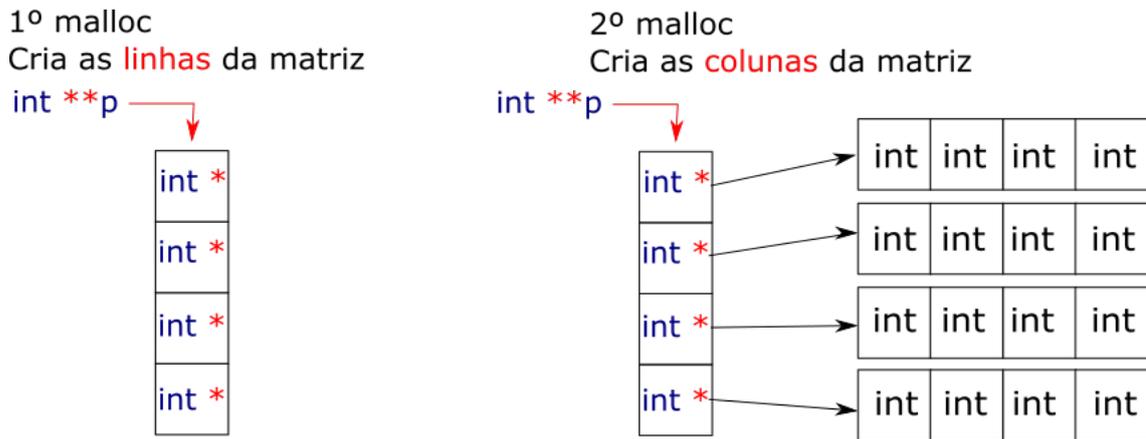


Figura 27 – Representação didática de alocação dinâmica de matriz

Fonte: Adaptado de (BACKES, 2013, p. 235)

será feito o primeiro **malloc()** que constrói as linhas, e no segundo laço, interno, será feito o segundo **malloc()** que constrói as colunas. Veja a seguir um exemplo.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int i, j, m, n;
7     int **matriz;
8     printf("Informe o numero de linhas: \n");
9     scanf("%d", &m);
10    printf("Informe o numero de colunas: \n");
11    scanf("%d", &n);
12    //primeiro malloc aloca as linhas
13    matriz = (int **)malloc(m * sizeof(int*));
14    for(i = 0; i < m; i++){
15        //segundo malloc aloca as colunas
16        matriz[i] = (int*) malloc(n * sizeof(int));
17
18        //acessando a matriz
19        for(j = 0; j < n; j++){
20            printf("Informe um valor: \n");
21            scanf("%d", &matriz[i][j]);
22        }
23    }
24 }
```

Na linha 7 foi declarado o ponteiro para ponteiro com nome de **matriz**, esse será responsável por alocar a memória da matriz. Entre as linhas 8 e 11 foram informados, em tempo de execução, as dimensões da matriz, o número de linhas foi armazenado na variável **m** e o número de colunas foi armazenado na variável **n**. Na linha 13 são alocadas as linhas da matriz, sendo que, na prática é um vetor de ponteiros, pois cada elemento desse vetor possui um ponteiro que será utilizado para alocar memória para as colunas. É iniciado o laço **for** na linha 14, neste laço, será alocada a memória para cada linha, ou seja, as colunas da linha, isso é realizado na linha 16. Na linha 19 é

iniciado o segundo laço **for** que é responsável pelo acesso aos valores da matriz.

### 10.4.1 Liberando a matriz da memória

No exemplo apresentado, a matriz não é liberada da memória, isso foi feito de propósito, pois para liberar uma matriz da memória é necessário um tratamento especial, pois como visto no código-fonte de exemplo, para cada linha da matriz alocada de forma dinâmica, tem-se outro vetor alocado dinamicamente, desta forma, para liberar esta estrutura da memória, é necessário realizar um trabalho de liberar cada uma destas estruturas que foram alocadas. Como fazer isso? Veja o exemplo adaptado.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int i, j, m, n;
7     int **matriz;
8     printf("Informe o numero de linhas: \n");
9     scanf("%d", &m);
10    printf("Informe o numero de colunas: \n");
11    scanf("%d", &n);
12    //primeiro malloc aloca as linhas
13    matriz = (int **)malloc(m * sizeof(int*));
14    for(i = 0; i < m; i++){
15        //segundo malloc aloca as colunas
16        matriz[i] = (int*) malloc(n * sizeof(int));
17
18        //acessando a matriz
19        for(j = 0; j < n; j++){
20            printf("Informe um valor: \n");
21            scanf("%d", &matriz[i][j]);
22        }
23    }
24    for (i = 0; i < m; i++)
25        free(matriz[i]);
26    free(matriz);
27 }
```

Este exemplo, é exatamente igual ao anterior, exceto pelas últimas linhas, entre a 24 e 26. Nestas linhas tem-se um laço **for** que passa por cada linha da matriz e libera as colunas da memória, note que, ao invocar a função **free()**, é informado o argumento **matriz[i]**, esse índice **i**, garante que cada linha será informada à função **free()**. Após liberar todas as colunas, as linhas, ou seja, o vetor de ponteiros, é liberado na linha 26.

## 10.5 Resumo da Aula

Neste aula foram discutidos importantes pontos sobre a alocação dinâmica de memória. Salientou-se a importância da alocação dinâmica para situações em que o número de elementos que pretende-se armazenar é desconhecido. Após isso, foram apresentadas as cinco funções que permitem ao programador trabalhar com alocação dinâmica de memória, **malloc()**, **calloc()**, **realloc()**, **free()** e **sizeof()**, sendo discutido sobre cada função com exemplos práticos de aplicação.

Além disso, foi discutido também sobre como alocar dinamicamente matrizes, neste tópico foram abordados vários aspectos que dependem de particularidades específicas em seu tratamento, como a necessidade de um laço para alocar memória para as colunas em cada linha da matriz e também a necessidade de laço para liberar a memória da matriz para uso.

## 10.6 Exercícios da Aula

Os exercícios desta lista foram Adaptados de [Backes \(2013, p. 238-239\)](#).

1. Escreva um programa que mostre o tamanho em byte que cada tipo de dados ocupa na memória: **char**, **int**, **float**, **double**.
2. Elabore um programa que leia do usuário o tamanho de um vetor a ser lido. Em seguida, faça a alocação dinâmica desse vetor. Por fim, leia o vetor do usuário e o imprima.
3. Faça um programa que leia um valor inteiro **N** não negativo. Se o valor de **N** for inválido, o usuário deverá digitar outro até que ele seja válido (ou seja, positivo). Em seguida, leia um vetor **V** contendo **N** posições de inteiros, em que cada valor deverá ser maior ou igual a 2. Esse vetor deverá ser alocado dinamicamente.
4. Faça uma função que retorne o ponteiro para um vetor de **N** elementos inteiros alocados dinamicamente. O vetor deve ser preenchido com valores de **0** a **N-1**.
5. Escreva uma função que receba um valor inteiro positivo **N** por parâmetro e retorne o ponteiro para um vetor de tamanho **N** alocado dinamicamente. Se **N** for negativo ou igual a zero, um ponteiro nulo deverá ser retornado.
6. Crie uma função que receba um texto e retorne o ponteiro para esse texto invertido.
7. Escreva uma função que receba como parâmetro dois vetores, **A** e **B**, de tamanho **N** cada. A função deve retornar o ponteiro para um vetor **C** de tamanho **N** alocado dinamicamente, em que  $C[i] = A[i] + B[i]$ .
8. Escreva uma função que receba como parâmetro dois vetores, **A** e **B**, de tamanho **N** cada. A função deve retornar o ponteiro para um vetor **C** de tamanho **N** alocado dinamicamente, em que  $C[i] = A[i] * B[i]$ .
9. Escreva um programa que aloque dinamicamente uma matriz de inteiros. As dimensões da matriz deverão ser lidas do usuário. Em seguida, escreva uma função que receba um valor e retorne **1**, caso o valor esteja na matriz, ou retorne **0**, no caso contrário.
10. Escreva um programa que leia um inteiro **N** e crie uma matriz alocada dinamicamente contendo **N** linhas e **N** colunas. Essa matriz deve conter o valor **0** na diagonal principal, o valor **1** nos elementos acima da diagonal principal e o valor **-1** nos elementos abaixo da diagonal principal. Veja a figura 29 para entender melhor o preenchimento da matriz.

0	1	1	1
-1	0	1	1
-1	-1	0	1
-1	-1	-1	0

Diagonal principal

Figura 28 – Formato de preenchimento da matriz para o exercício



## Exercícios Resolvidos da Aula 1

1. Faça um programa em C que imprima o seu nome.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     char nome[100];
7     printf("Digite seu nome: \n");
8     scanf("%s", &nome);
9     printf("\n%s", nome);
10 }
```

2. Faça um programa em C que imprima o produto dos valores 30 e 27.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int resultado;
7     resultado = 30 * 27;
8     printf("Resultado: %d", resultado);
9 }
```

3. Faça um programa em C que imprima a média aritmética entre os números 5, 8, 12.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     float resultado;
7     resultado = (5 + 8 + 12) / 3;
8     printf("Resultado: %f", resultado);
9 }
```

4. Faça um programa em C que leia e imprima um número inteiro.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int n;
7     printf("Digite o numero: ");
8     scanf("%d",&n);
9     printf("Numero digitado: %d", n);
10 }
```

5. Faça um programa em C que leia dois números reais e os imprima.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     float n, n2;
7
8     printf("Digite o numero 1: ");
9     scanf("%f",&n);
10    printf("Digite o numero 2: ");
11    scanf("%f",&n2);
12    printf("Numeros reais: %f %f", n, n2);
13 }
```

6. Faça um programa em C que leia um número inteiro e imprima o seu antecessor e o seu sucessor.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     float n;
7     printf("Digite o numero: ");
8     scanf("%f",&n);
9     printf("Antecessor: %f e Sucessor: %f", n-1, n+1);
10 }
```

7. Faça um programa em C que leia o nome o endereço e o telefone de um cliente e ao final, imprima esses dados.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     char nome[100], telefone[100], endereco[100];
7     printf("Digite seu nome: ");
8     scanf("%s", &nome);
```

```
9     printf("Digite seu telefone: ");
10     scanf("%s", &telefone);
11     printf("Digite seu endereco: ");
12     scanf("%s", &endereco);
13     printf("\nDados - Nome:%s, Telefone: %s, Enderenco: %s",nome, telefone,
14           endereco);
15 }
```

8. Faça um programa em C que leia dois números inteiros e imprima a subtração deles.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int n, n2, subtracao;
7     printf("Digite o numero 1: ");
8     scanf("%d", &n);
9     printf("Digite o numero 2: ");
10    scanf("%d", &n2);
11    subtracao = n - n2;
12    printf("Resultado: %d", subtracao);
13 }
```

9. Faça um programa em C que leia um número real e imprima  $\frac{1}{4}$  deste número.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     float n, resultado;
7     printf("Digite o numero: ");
8     scanf("%f", &n);
9     resultado = n / 4;
10    printf("Resultado: %f", resultado);
11 }
```

10. Faça um programa em C que leia três números reais e calcule a média aritmética destes números. Ao final, o programa deve imprimir o resultado do cálculo.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     float n1, n2, n3, media;
7     printf("Digite o numero 1: ");
8     scanf("%f",&n1);
9     printf("Digite o numero 2: ");
10    scanf("%f",&n2);
11    printf("Digite o numero 3: ");
```

```
12     scanf("%f",&n3);
13     media = (n1+n2+n3) / 3;
14     printf("Media: %f ", media);
15 }
```

11. Faça um programa em C que leia dois números reais e calcule as quatro operações básicas entre estes dois números, adição, subtração, multiplicação e divisão. Ao final, o programa deve imprimir os resultados dos cálculos.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     float n1, n2, resultado;
7     printf("Digite o numero 1: ");
8     scanf("%f", &n1);
9     printf("Digite o numero 2: ");
10    scanf("%f", &n2);
11    resultado = n1 + n2;
12    printf("\nSoma: %f", resultado);
13    resultado = n1 - n2;
14    printf("\nSubtracao: %f", resultado);
15    resultado = n1 * n2;
16    printf("\nMultiplicacao: %f", resultado);
17    resultado = n1 / n2;
18    printf("\nDivisao: %f", resultado);
19 }
```

12. Faça um programa em C que leia um número real e calcule o quadrado deste número. Ao final, o programa deve imprimir o resultado do cálculo.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     float num, resultado;
7     printf("Digite o numero: ");
8     scanf("%f", &num);
9     resultado = num * num;
10    printf("Quadrado do numero: %f", resultado);
11 }
```

13. Faça um programa em C que leia o saldo de uma conta poupança e imprima o novo saldo, considerando um reajuste de 2%.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     float saldo, novoSaldo;
```

```
7     printf("Digite o saldo:");
8     scanf("%f", &saldo);
9     novoSaldo = saldo + saldo * 0.02;
10    printf("Saldo com reajuste: %f", novoSaldo);
11 }
```

14. Faça um programa em C que leia a base e a altura de um retângulo e imprima o perímetro (base + altura) e a área (base \* altura).

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     float base, altura, perimetro, area;
7     printf("Digite a base: ");
8     scanf("%f", &base);
9     printf("Digite a altura: ");
10    scanf("%f", &altura);
11    perimetro = base + altura;
12    area = base * altura;
13    printf("Perimetro: %f \n", perimetro);
14    printf("Area: %f \n", area);
15 }
```

15. Faça um programa em C que leia o valor de um produto, o percentual do desconto desejado e imprima o valor do desconto e o valor do produto subtraindo o desconto.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     float vlrProd, percDesc, vlrDesc;
7     printf("Digite o valor do produto: ");
8     scanf("%f", &vlrProd);
9     printf("Digite o desconto: ");
10    scanf("%f", &percDesc);
11    vlrDesc = vlrProd - (vlrProd * (percDesc / 100));
12    printf("Produto com desconto: %f", vlrDesc);
13 }
```

16. Faça um programa em C que calcule o reajuste do salário de um funcionário. Para isso, o programa deverá ler o salário atual do funcionário e ler o percentual de reajuste. Ao final imprimir o valor do novo salário.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     float salario, percReajuste, vlrReajuste;
```

```

7     printf("Digite o valor do salario do funcionario: ");
8     scanf("%f", &salario);
9     printf("Digite o percentual de reajuste: ");
10    scanf("%f", &percReajuste);
11    vlrReajuste = salario + (salario * (percReajuste / 100));
12    printf("Produto com desconto: %f", vlrReajuste);
13 }

```

17. Faça um programa em C que calcule a conversão entre graus centígrados e Fahrenheit. Para isso, leia o valor em centígrados e calcule com base na fórmula a seguir. Após calcular o programa deve imprimir o resultado da conversão.

$$F = \frac{9 * C + 160}{5} \quad (\text{A.1})$$

Em que:

- F = Graus em Fahrenheit
- C = Graus centígrados

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     float gCentigrados, gFahrenheit;
7     printf("Digite o valor em graus centigrados: ");
8     scanf("%f", &gCentigrados);
9     gFahrenheit = (9 * gCentigrados + 160) / 5;
10    printf("Temperatura em fahrenheit: %f", gFahrenheit);
11 }

```

18. Faça um programa em C que calcule a quantidade de litros de combustível consumidos em uma viagem, sabendo-se que o carro tem autonomia de 12 km por litro de combustível. O programa deverá ler o tempo decorrido na viagem e a velocidade média e aplicar as fórmulas:

$$D = T * V \quad (\text{A.2})$$

$$L = \frac{D}{12} \quad (\text{A.3})$$

Em que:

- D = Distância percorrida
- T = Tempo decorrido em horas
- V = Velocidade média
- L = Litros de combustível consumidos

Ao final, o programa deverá imprimir a distância percorrida e a quantidade de litros consumidos na viagem.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     float tempoDecorrido, velocidadeMedia, distanciaPercorrida,
7         combustivel;
8     printf("Digite o tempo de viagem em horas: ");
9     scanf("%f",&tempoDecorrido);
10    printf("Digite a velocidade media: ");
11    scanf("%f",&velocidadeMedia);
12    distanciaPercorrida = tempoDecorrido * velocidadeMedia;
13    combustivel = distanciaPercorrida / 12;
14    printf("Quantidade de litros consumidos: %f", combustivel);
15 }

```

19. Faça um programa em C que calcule o valor de uma prestação em atraso. Para isso, o programa deve ler o valor da prestação vencida, a taxa periódica de juros e o período de atraso. Ao final, o programa deve imprimir o valor da prestação atrasada, o período de atraso, os juros que serão cobrados pelo período de atraso, o valor da prestação acrescido dos juros. Considere juros simples.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int periodoAtraso;
7     float valorVencido, taxaJuros, juros, novoValor;
8     printf("Digite o valor da prestacao vencida: ");
9     scanf("%f", &valorVencido);
10    printf("Taxa de juros: ");
11    scanf("%f", &taxaJuros);
12    printf("Periodo de atraso: ");
13    scanf("%d", &periodoAtraso);
14    juros = ((valorVencido * (taxaJuros / 100)) * periodoAtraso);
15    novoValor = valorVencido + juros;
16    printf("Valor prestacao: %f \n", valorVencido);
17    printf("Periodo de atraso: %d \n", periodoAtraso);
18    printf("Juros a ser cobrados: %f \n", juros);
19    printf("Valor da prestacao com juros: %.2f", novoValor);
20 }

```

20. Faça um programa em C que efetue a apresentação do valor da conversão em real (R\$) de um valor lido em dólar (US\$). Para isso, será necessário também ler o valor da cotação do dólar.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     float cotDolar, vlrDolar, conversao;

```

```
7
8     printf("Digite o valor em dolar: ");
9     scanf("%f", &vlrDolar);
10    printf("Digite a cotacao do dolar: ");
11    scanf("%f", &cotDolar);
12    conversao = vlrDolar * cotDolar;
13    printf("Conversao em reais: %.2f", conversao);
14 }
```

## Exercícios Resolvidos da Aula 2

1. Faça um programa em C que leia dois valores numéricos inteiros e efetue a adição, caso o resultado seja maior que 10, apresentá-lo.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int n1, n2, adicao;
7     printf("Digite o numero 1: ");
8     scanf("%d",&n1);
9     printf("Digite o numero 2: ");
10    scanf("%d",&n2);
11    adicao = n1 + n2;
12    if (adicao > 10)
13        printf("%d", adicao);
14 }
```

2. Faça um programa em C que leia dois valores inteiros e efetue a adição. Caso o valor somado seja maior que 20, este deverá ser apresentado somando-se a ele mais 8, caso o valor somado seja menor ou igual a 20, este deverá ser apresentado subtraindo-se 5.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int n1, n2, adicao;
7     printf("Digite o numero 1: ");
8     scanf("%d",&n1);
9     printf("Digite o numero 2: ");
10    scanf("%d",&n2);
11    adicao = n1 + n2;
12    if (adicao > 20) {
13        adicao += 8;
14    }
15    else {
```

```
16     adicao -= 5;
17     }
18     printf("%d", adicao);
19 }
```

3. Faça um programa que leia um número e imprima uma das duas mensagens: "É múltiplo de 3" ou "Não é múltiplo de 3".

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int num;
7     printf("Digite o numero: ");
8     scanf("%d", &num);
9     if ((num % 3) == 0)
10        printf("E multiplo de 3");
11    else
12        printf("Nao e Multiplo de 3");
13 }
```

4. Faça um programa que leia um número e informe se ele é ou não divisível por 5.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int num;
7     printf("Digite o numero: ");
8     scanf("%d", &num);
9     if ((num % 5) == 0)
10        printf("E divisivel por 5");
11    else
12        printf("Nao e divisivel por 5");
13 }
```

5. Faça um programa em C que leia um número e informe se ele é divisível por 3 e por 7.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int num;
7     printf("Digite o numero: ");
8     scanf("%d", &num);
9     if ((num % 3) == 0 && (num % 7) == 0)
10        printf("Divisivel por 3 e por 7");
11    else
12        printf("Nao e divisivel por 3 e por 7");
```

```
13 }
```

6. A prefeitura do Rio de Janeiro abriu uma linha de crédito para os funcionários estatutários. O valor máximo da prestação não poderá ultrapassar 30% do salário bruto. Faça um programa em linguagem C que permita entrar com o salário bruto e o valor da prestação e informar se o empréstimo pode ou não ser concedido.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int salario, prestacao;
7     printf("Digite o salario bruto: ");
8     scanf("%d",&salario);
9     printf("Digite o valor da prestacao: ");
10    scanf("%d",&prestacao);
11    if (prestacao <= (salario * 0.3))
12        printf("Emprestimo consedido");
13    else
14        printf("Emprestimo nao consedido");
15 }
```

7. Faça um programa em C que leia um número e indique se o número está compreendido entre 20 e 50 ou não.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int num;
7     printf("Digite o numero: ");
8     scanf("%d",&num);
9     if (num >= 20 && num <= 50)
10        printf("Numero entre 20 e 50");
11    else
12        printf("Numero menor que 20 ou maior que 50");
13 }
```

8. Faça um programa que leia um número e imprima uma das mensagens: "Maior do que 20", "Igual a 20" ou "Menor do que 20".

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int num;
7     printf("Digite o numero: ");
8     scanf("%d", &num);
9     if (num > 20)
10        printf("Numero maior que 20");
```

```
11     else if (num == 20)
12         printf("Numero igual a 20");
13     else
14         printf("Numero menor que 20");
15 }
```

9. Faça um programa em C que permita entrar com o ano de nascimento da pessoa e com o ano atual. O programa deve imprimir a idade da pessoa. Não se esqueça de verificar se o ano de nascimento informado é válido.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void main()
5  {
6      int nascimento, anoAtual;
7      printf("Digite o ano de nascimento: \n");
8      scanf("%d", &nascimento);
9      printf("Digite o ano atual: \n");
10     scanf("%d", &anoAtual);
11     if (nascimento > 0 && nascimento <= anoAtual) {
12         printf("Sua idade: %d", anoAtual - nascimento);
13     }
14     else
15         printf("Data de nascimento invalida");
16 }
```

10. Faça um programa em C que leia três números inteiros e imprima os três em ordem crescente.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void main()
5  {
6      int n1, n2, n3;
7      printf("Digite o primeiro numero: ");
8      scanf("%d",&n1);
9      printf("Digite o segundo numero: ");
10     scanf("%d",&n2);
11     printf("Digite o terceiro numero: ");
12     scanf("%d",&n3);
13     if (n1 < n2 && n1 < n3) {
14         if (n2 < n3)
15             printf("%d, %d, %d", n1, n2, n3);
16         else
17             printf("%d, %d, %d", n1, n3, n2);
18     }
19     else if (n2 < n1 && n2 < n3) {
20         if (n1 < n3)
21             printf("%d, %d, %d", n2, n1, n3);
22         else
23             printf("%d, %d, %d", n2, n3, n1);
24     }
```

```
25     else {
26         if (n2 < n1)
27             printf("%d, %d, %d", n3, n2, n1);
28         else
29             printf("%d, %d, %d", n3, n1, n2);
30     }
31 }
```

11. Faça um programa que leia 3 números e imprima o maior deles.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int n1, n2, n3;
7     printf("Digite o primeiro numero: ");
8     scanf("%d",&n1);
9     printf("Digite o segundo numero: ");
10    scanf("%d",&n2);
11    printf("Digite o terceiro numero: ");
12    scanf("%d",&n3);
13
14    if (n1 > n2 && n1 > n3) {
15        printf("Maior numero: %d", n1);
16    }
17    else if (n2 > n1 && n2 > n3) {
18        printf("Maior numero: %d", n2);
19    }
20    else {
21        printf("Maior numero: %d", n3);
22    }
23 }
```

12. Faça um programa que leia a idade de uma pessoa e informe:

- Se é maior de idade
- Se é menor de idade
- Se é maior de 65 anos

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int idade;
7     printf("Digite a idade: ");
8     scanf("%d",&idade);
9     if (idade >= 65)
10        printf("Maior que 65");
11    else if (idade >= 18)
12        printf("Maior de idade");
13    else
```

```

14     printf("Menor de idade");
15 }

```

13. Faça um programa que permita entrar com o nome, a nota da prova 1 e a nota da prova 2 de um aluno. O programa deve imprimir o nome, a nota da prova 1, a nota da prova 2, a média das notas e uma das mensagens: "Aprovado", "Reprovado" ou "em Prova Final" (a média é 7 para aprovação, menor que 3 para reprovação e as demais em prova final).

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void main()
5  {
6      float p1, p2, media;
7      char nome[30];
8      printf("Nome do aluno: ");
9      scanf("%s", &nome);
10     printf("Digite a nota da prova 1: ");
11     scanf("%f",&p1);
12     printf("Digite a nota da prova 2: ");
13     scanf("%f",&p2);
14     media = (p1 + p2) / 2;
15     printf("-----Dados-----\n");
16     printf("Aluno: %s \n", nome);
17     printf("Notas - P1: %f P2: %f \n", p1, p2);
18     printf("Media: %f \n", media);
19     if (media >= 7)
20         printf("Aprovado");
21     else if (media >= 3)
22         printf("Prova final");
23     else
24         printf("\nReprovado");
25 }

```

14. Faça um programa que permita entrar com o salário de uma pessoa e imprima o desconto do INSS segundo a tabela seguir:

Salário	Faixa de Desconto
Menor ou igual à R\$600,00	Isento
Maior que R\$600,00 e menor ou igual a R\$1200,00	20%
Maior que R\$1200,00 e menor ou igual a R\$2000,00	25%
Maior que R\$2000,00	30%

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void main()
5  {
6      int salario;
7      printf("Digite o salario: ");
8      scanf("%d",&salario);

```

```

9     if (salario <= 600)
10        printf("Isento de INSS");
11     else if (salario <= 1200)
12        printf("20%: %.2f", salario * 0.2);
13     else if (salario <= 2000)
14        printf("25%: %.2f", salario * 0.25);
15     else
16        printf("30%: %.2f", salario * 0.3);
17 }

```

15. Um comerciante comprou um produto e quer vendê-lo com um lucro de 45% se o valor da compra for menor que R\$20,00, caso contrário, o lucro será de 30%. Faça um programa em C que leia o valor do produto e imprima o valor da venda.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     float vlrProduto, vlrVenda;
7     printf("Digite o valor do produto: ");
8     scanf("%f", &vlrProduto);
9     if (vlrProduto < 20) {
10        vlrVenda = vlrProduto + (vlrProduto * 0.45);
11    }
12    else {
13        vlrVenda = vlrProduto + (vlrProduto * 0.3);
14    }
15    printf("Valor do produto para venda: %.2f \n", vlrVenda);
16 }

```

16. A confederação brasileira de natação irá promover eliminatórias para o próximo mundial. Faça um programa em C que receba a idade de um nadador e imprima a sua categoria segundo a tabela a seguir:

Categoria	Idade
Infantil A	5 - 7 anos
Infantil B	8 - 10 anos
Juvenil A	11 - 13 anos
Juvenil B	14 - 17 anos
Sênior	maiores de 18 anos

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int idade;
7     printf("Digite a idade: ");
8     scanf("%d", &idade);
9     if (idade <= 4)
10        printf("Idade nao permitida");

```

```

11     else {
12         if (idade <= 7)
13             printf("Infantil A");
14             else if (idade <= 10)
15                 printf("Infantil B");
16         else if (idade <= 13)
17             printf("Juvenil A");
18         else if (idade <= 17)
19             printf("Juvenil B");
20         else
21             printf("Senior");
22     }
23 }

```

17. Depois da liberação do governo para as mensalidades dos planos de saúde, as pessoas começaram a fazer pesquisas para descobrir um bom plano, não muito caro. Um vendedor de um plano de saúde apresentou a tabela a seguir. Faça um programa que entre com o nome e a idade de uma pessoa e imprima o nome e o valor que ela deverá pagar.

Idade	Valor
Até 10 anos	R\$30,00
Acima de 10 até 29 anos	R\$60,00
Acima de 29 até 45 anos	R\$120,00
Acima de 45 até 59 anos	R\$150,00
Acima de 59 até 65 anos	R\$250,00
Maior que 65 anos	R\$400,00

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void main()
5  {
6      char nome[100];
7      int idade;
8      printf("Digite o nome: ");
9      scanf("%s", &nome);
10     printf("Digite a idade: ");
11     scanf("%d", &idade);
12     if (idade <= 10)
13         printf("Valor: 30 reais");
14     else if (idade <= 29)
15         printf("Valor: 60 reais");
16     else if (idade <= 45)
17         printf("Valor: 120 reais");
18     else if (idade <= 59)
19         printf("Valor: 150 reais");
20     else if (idade <= 65)
21         printf("Valor: 250 reais");
22     else
23         printf("Valor: 400 reais");
24 }

```

18. Faça um programa que leia um número inteiro entre 1 e 12 e escreva o mês correspondente. Caso o usuário digite um número fora desse intervalo, deverá aparecer uma mensagem informando que não existe mês com este número. Utilize o **switch** para este problema.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int mes;
7     printf("Digite o mes: ");
8     scanf("%d",&mes);
9     switch (mes) {
10    case 1:
11        printf("Janeiro \n");
12        break;
13    case 2:
14        printf("Fevereiro \n");
15        break;
16    case 3:
17        printf("Marco \n");
18        break;
19    case 4:
20        printf("Abril \n");
21        break;
22    case 5:
23        printf("Maio \n");
24        break;
25    case 6:
26        printf("Junho \n");
27        break;
28    case 7:
29        printf("Julho \n");
30        break;
31    case 8:
32        printf("Agosto \n");
33        break;
34    case 9:
35        printf("Setembro \n");
36        break;
37    case 10:
38        printf("Outubro \n");
39        break;
40    case 11:
41        printf("Novembro \n");
42        break;
43    case 12:
44        printf("Dezembro \n");
45        break;
46    default:
47        printf("Mes invalido! \n");
48    }
49 }
```

19. Em um campeonato nacional de arco-e-flecha, tem-se equipes de três jogadores para cada estado. Sabendo-se que os arqueiros de uma equipe não obtiveram o mesmo número de pontos, criar um programa em C que informe se uma equipe foi classificada, de acordo com a seguinte especificação:

- Ler os pontos obtidos por cada jogador da equipe;
- Mostrar esses valores em ordem decrescente;
- Se a soma dos pontos for maior do que 100, imprimir a média aritmética entre eles, caso contrário, imprimir a mensagem "Equipe desclassificada".

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     float n1, n2, n3;
7     printf("Digite os pontos do primeiro atleta: ");
8     scanf("%f",&n1);
9     printf("Digite os pontos do segundo atleta: ");
10    scanf("%f",&n2);
11    printf("Digite os pontos do terceiro atleta: ");
12    scanf("%f",&n3);
13    if (n1 < n2 && n1 < n3) {
14        if (n2 < n3)
15            printf("%f, %f, %f", n1, n2, n3);
16        else
17            printf("%f, %f, %f", n1, n3, n2);
18    }
19    else if (n2 < n1 && n2 < n3) {
20        if (n1 < n3)
21            printf("%f, %f, %f", n2, n1, n3);
22        else
23            printf("%f, %f, %f", n2, n3, n1);
24    }
25    else {
26        if (n2 < n1)
27            printf("%f, %f, %f", n3, n2, n1);
28        else
29            printf("%f, %f, %f", n3, n1, n2);
30    }
31    float media = (n1 + n2 + n3) / 3;
32    if ((n1 + n2 + n3) >= 100)
33        printf("\n%.2f", media);
34    else
35        printf("\nEquipe desclassificada");
36 }
```

20. O banco XXX concederá um crédito especial com juros de 2% aos seus clientes de acordo com o saldo médio no último ano. Faça um programa que leia o saldo médio de um cliente e calcule o valor do crédito de acordo com a tabela a seguir. O programa deve imprimir uma mensagem informando o saldo médio e o valor de crédito.

Saldo Médio	Percentual
de 0 a 500	nenhum crédito
de 501 a 1000	30% do valor do saldo médio
de 1001 a 3000	40% do valor do saldo médio
acima de 3001	50% do valor do saldo médio

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int saldo;
7     printf("Digite o saldo: ");
8     scanf("%d",&saldo);
9     if (saldo <= 500)
10        printf("Nenhum credito");
11    else if (saldo <= 1000)
12        printf("Saldo: %d - Credito: %f", saldo, saldo * 0.3);
13    else if (saldo <= 3000)
14        printf("Saldo: %d - Credito: %f", saldo, saldo * 0.4);
15    else
16        printf("Saldo: %d - Credito: %f", saldo, saldo * 0.5);
17 }

```

21. A biblioteca de uma Universidade deseja fazer um programa que leia o nome do livro que será emprestado, o tipo de usuário (professor ou aluno) e possa imprimir um recibo conforme mostrado a seguir. Considerar que o professor tem dez dias para devolver o livro e o aluno só três dias.

- Nome do livro:
- Tipo de usuário:
- Total de dias:

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int tipo;
7     char livro[100];
8     printf("Digite o nome do livro: ");
9     scanf("%s", &livro);
10    printf("Digite o tipo de usuario <1-Professor / 2-Aluno>: ");
11    scanf("%d", &tipo);
12    printf("Nome do livro: %s\n", livro);
13    if (tipo == 1) {
14        printf("Tipo de usuario: Professor \nTotal de Dias: 10");
15    }
16    else {
17        printf("Tipo de usuario: Aluno \nTotal de Dias: 3");
18    }
19 }

```

22. Construa um programa que leia o percurso em quilômetros, o tipo do carro e informe o consumo estimado de combustível, sabendo-se que um carro tipo C faz 12 km com um litro de gasolina, um tipo B faz 9 km e o tipo C, 8 km por litro.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     char tipo;
7     float percurso, consumo;
8     printf("Digite o tipo de carro <A, B ou C>: ");
9     scanf("%c", &tipo);
10    printf("Digite o percurso em KM: ");
11    scanf("%f", &percurso);
12    switch (tipo) {
13    case 'A':
14        consumo = percurso / 12;
15        printf("Consumo estimado: %.2f", consumo);
16        break;
17    case 'B':
18        consumo = percurso / 9;
19        printf("Consumo estimado: %.2f", consumo);
20        break;
21    case 'C':
22        consumo = percurso / 8;
23        printf("Consumo estimado: %.2f", consumo);
24        break;
25    default:
26        printf("Tipo de carro invalido!");
27    }
28 }

```

23. Crie um programa que informe a quantidade total de calorias de uma refeição a partir da escolha do usuário que deverá informar o prato, a sobremesa, e bebida conforme a tabela a seguir.

Prato	Sobremesa	Bebida
Vegetariano 180cal	Abacaxi 75cal	Chá 20cal
Peixe 230cal	Sorvete diet 110cal	Suco de laranja 70cal
Frango 250cal	Mousse diet 170cal	Suco de melão 100cal
Carne 350cal	Mousse chocolate 200cal	Refrigerante diet 65cal

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int calorias = 0;
7     int prato, sobremesa, bebida;
8     printf("\nPrato");
9     printf("\n1 - Vegetariano");
10    printf("\n2 - Peixe");
11    printf("\n3 - Frango");

```

```
12     printf("\n4 - Carne");
13     printf("\nDigite a opcao: ");
14     scanf("%d",&prato);
15     printf("\nSobremesa");
16     printf("\n1 - Abacaxi");
17     printf("\n2 - Sorvete diet");
18     printf("\n3 - Mousse diet");
19     printf("\n4 - Mousse chocolate");
20     printf("\nDigite a opcao: ");
21     scanf("%d",&sobremesa);
22     printf("\nBebida");
23     printf("\n1 - Cha");
24     printf("\n2 - Suco de laranja");
25     printf("\n3 - Suco de melao");
26     printf("\n4 - Refrigerante diet");
27     printf("\nDigite a opcao: ");
28     scanf("%d",&bebida);
29     switch (prato) {
30     case 1:
31         calorias += 180;
32         break;
33     case 2:
34         calorias += 230;
35         break;
36     case 3:
37         calorias += 250;
38         break;
39     case 4:
40         calorias += 350;
41         break;
42     }
43     switch (sobremesa) {
44     case 1:
45         calorias += 75;
46         break;
47     case 2:
48         calorias += 110;
49         break;
50     case 3:
51         calorias += 170;
52         break;
53     case 4:
54         calorias += 200;
55         break;
56     }
57     switch (bebida) {
58     case 1:
59         calorias += 20;
60         break;
61     case 2:
62         calorias += 70;
63         break;
64     case 3:
65         calorias += 100;
66         break;
```

```

67     case 4:
68         calorias += 65;
69         break;
70     }
71     printf("Total de calorias: %d",calorias);
72 }

```

24. A polícia rodoviária resolveu fazer cumprir a lei e vistoriar veículos para cobrar dos motoristas o DUT. Sabendo-se que o mês em que o emplacamento do carro deve ser renovado é determinado pelo último número da placa do mesmo, faça um programa que, a partir da leitura da placa do carro, informe o mês em que o emplacamento deve ser renovado.

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void main()
5  {
6      int placa, milhar, centena, dezena;
7      printf("Digite o numero da placa do carro no formato <9999>: ");
8      scanf("%d", &placa);
9      milhar = placa / 1000;
10     placa = placa - (milhar * 1000);
11     centena = placa / 100;
12     placa = placa - (centena * 100);
13     dezena = placa / 10;
14     placa = placa - (dezena * 10);
15     printf("Mes: %d", placa);
16 }

```

25. A prefeitura contratou uma firma especializada para manter os níveis de poluição considerados ideais para um país do 1º mundo. As indústrias, maiores responsáveis pela poluição, foram classificadas em três grupos. Sabendo-se que a escala utilizada varia de 0,05 e que o índice de poluição aceitável é até 0,25, fazer um programa que possa imprimir intimações de acordo com o índice e a tabela a seguir:

Índice	Indústrias que receberão intimação
0,3	1º grupo
0,4	1º e 2º grupos
0,5	1º, 2º e 3º grupos

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void main()
5  {
6      float indice;
7      printf("Digite o indice de poluicao: \n");
8      scanf("%f", &indice);
9      if (indice >= 0.5) {

```

```
10     printf("Suspender atividades das industrias dos grupos 1, 2 e 3 \n"
11           );
12 }
13 else if (indice >= 0.4) {
14     printf("Suspender atividades das industrias dos grupos 1 e 2 \n");
15 }
16 else if (indice >= 0.3) {
17     printf("Suspender atividades das industrias do grupo 1 \n");
18 }
19 else {
20     printf("Indice de poluicao aceitavel para todos os grupos \n");
21 }
```



## Exercícios Resolvidos da Aula 3

1. Faça um programa em C que imprima todos os números de 1 até 100.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int i;
7     for (i=1; i<=100; i++)
8         printf("%d \n", i);
9 }
```

2. Faça um programa que imprima todos os números pares de 100 até 1.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int i;
7     for (i=100; i>=1; i--) {
8         if (i%2 == 0)
9             printf("%d \n", i);
10    }
11 }
```

3. Faça um programa que imprima os múltiplos de 5, no intervalo de 1 até 500.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int i;
7     for (i=1; i<=500; i++) {
8         if (i%5 == 0)
9             printf("%d \n", i);
10    }
11 }
```

```

10     }
11 }

```

4. Faça um programa em C que permita entrar com o nome, a idade e o sexo de 20 pessoas. O programa deve imprimir o nome da pessoa se ela for do sexo masculino e tiver mais de 21 anos.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     char nome[100];
7     int sexo, idade, i;
8
9     for (i=0; i<20; i++)
10    {
11        printf("Digite o nome: ");
12        fflush(stdin);
13        fgets(nome, 100, stdin);
14        printf("Digite a idade: ");
15        fflush(stdin);
16        scanf("%d", &idade);
17        printf("Digite o sexo <1-M / 2-F>: ");
18        fflush(stdin);
19        scanf("%d", &sexo);
20        if (sexo == 1 && idade > 21)
21            printf("Nome: %s \n", nome);
22    }
23 }

```

5. Sabendo-se que a unidade lógica e aritmética calcula o produto através de somas sucessivas, crie um programa que calcule o produto de dois números inteiros lidos. Suponha que os números lidos sejam positivos e que o multiplicando seja menor do que o multiplicador.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int i, multiplicando, multiplicador, soma=0;
7
8     printf("Digite o multiplicando: ");
9     scanf("%d", &multiplicando);
10    printf("Digite o multiplicador: ");
11    scanf("%d", &multiplicador);
12    if (multiplicando < 0)
13        printf("Numero de multiplicando invalido \n");
14    else if (multiplicador < 0)
15        printf("Numero de multiplicador invalido \n");
16    else if (multiplicando > multiplicador)
17        printf("Multiplicando deve ser menor que multiplicador \n");
18    else {

```

```

19     for (i=1; i <= multiplicador; i++) {
20         soma += multiplicando;
21     }
22     printf("Resultado: %d \n", soma);
23 }
24 }

```

6. Crie um programa em C que imprima os 20 primeiros termos da série de Fibonacci.

**Observação:** os dois primeiros termos desta série são 1 e 1 e os demais são gerados a partir da soma dos anteriores. Exemplo:

- $1 + 1 = 2$ , terceiro termo;
- $1 + 2 = 3$ , quarto termo, etc.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int i, fib=1, fib2=1, aux;
7
8     printf("%d \n", fib);
9     printf("%d \n", fib2);
10
11    for (i=2; i<20; i++) {
12        aux = fib2;
13        fib2 = fib + fib2;
14        printf("%d \n", fib2);
15        fib = aux;
16    }
17 }

```

7. Crie um programa em linguagem C que permita entrar com o nome, a nota da prova 1 e da prova 2 de 15 alunos. Ao final, imprimir uma listagem, contendo: nome, nota da prova 1, nota da prova 2, e média das notas de cada aluno. Ao final, imprimir a média geral da turma.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     char nome[100];
7     int p1, p2, i;
8     float media=0, mediafinal=0;
9
10    for (i=1; i<=5; i++) {
11        printf("Digite o nome: ");
12        fflush(stdin);
13        fgets(nome, 100, stdin);
14        printf("Digite a nota da prova 1: ");
15        scanf("%d", &p1);

```

```

16     printf("Digite a nota da prova 2: ");
17     scanf("%d", &p2);
18     printf("Nome: %s", nome);
19     printf("Notas - P1: %d \t P2: %d \n", p1, p2);
20     media = (p1 + p2) / 2;
21     printf("Media de notas: %.2f \n", media);
22     mediafinal += media;
23 }
24 mediafinal = mediafinal / (i-1);
25 printf("Media de notas dos alunos: %f \n", mediafinal);
26 }

```

8. Faça um programa que permita entrar com o nome e o salário bruto de 10 pessoas. Após ler os dados, imprimir o nome e o valor da alíquota do imposto de renda calculado conforme a tabela a seguir:

Salário	IRRF
Salário menor que R\$1300,00	Isento
Salário maior ou igual a R\$1300,00 e menor que R\$2300,00	10% do salário bruto
Salário maior ou igual a R\$2300,00	15% do salário bruto

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void main()
5  {
6      int i;
7      char nome[100];
8      float salario, aliquota;
9
10     for (i=1; i<=10; i++) {
11         printf("Digite o nome: \n");
12         fflush(stdin);
13         fgets(nome, 100, stdin);
14         printf("Digite o salario: \n");
15         scanf("%f", &salario);
16         if (salario <= 1300) {
17             printf("Isento de imposto \n");
18         }
19         else {
20             if (salario <= 2300) {
21                 aliquota = (salario * 10) / 100;
22             }
23             else {
24                 aliquota = (salario * 15) / 100;
25             }
26             printf("Aliquota: %f \n", aliquota);
27         }
28     }
29 }

```

9. No dia da estréia do filme "Procurando Dory", uma grande emissora de TV realizou uma pesquisa logo após o encerramento do filme. Cada espectador respondeu

a um questionário no qual constava sua idade e a sua opinião em relação ao filme: excelente - 3; bom - 2; regular - 1. Criar um programa que receba a idade e a opinião de 20 espectadores, calcule e imprima:

- A média das idades das pessoas que responderam excelente;
- A quantidade de pessoas que responderam regular;
- A percentagem de pessoas que responderam bom entre todos os espectadores analisados.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int idade, bom=0, regular=0, excelente=0, opcao, i;
7     float mediaIdade=0, percBom;
8
9     for (i=1; i<=20; i++) {
10        printf("Digite a idade: ");
11        scanf("%d", &idade);
12        printf("Responda sua opiniao sobre o filme: \n");
13        printf("1 - regular \t");
14        printf("2 - bom \t");
15        printf("3 - excelente \n");
16        scanf("%d", &opcao);
17        if (opcao == 1)
18            regular++;
19        else if (opcao == 2)
20            bom++;
21        else {
22            mediaIdade += idade;
23            excelente++;
24        }
25    }
26    mediaIdade = mediaIdade / excelente;
27    printf("Media de idades das pessoas que responderam excelente: %f \n",
28        mediaIdade);
29    printf("Quantidade de pessoas que responderam regular: %d \n", regular
30    );
31    percBom = bom;
32    percBom = percBom / (i-1);
33    printf("Porcentagem de pessoas que responderam bom: %f \n", percBom);
34 }

```

10. Em um campeonato Europeu de Volleyball, se inscreveram 30 países. Sabendo-se que na lista oficial de cada país consta, além de outros dados, peso e idade de 12 jogadores, crie um programa que apresente as seguintes informações:

- O peso médio e a idade média de cada um dos times;
- O atleta mais pesado de cada time;
- O atleta mais jovem de cada time;

- O peso médio e a idade média de todos os participantes.

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void main()
5  {
6      int i, j, peso, idade, pesado, jovem;
7      float pesoMedio, idadeMedia=0, pesoMedioTotal=0, idadeMediaTotal=0;
8
9      for (i=1; i<=30; i++) {
10         pesoMedio = 0;
11         idadeMedia = 0;
12         pesado = 0;
13         jovem = 0;
14         printf("Informacoes do time: %d \n", i);
15         for (j=1; j<=12; j++) {
16             printf("Digite o peso: ");
17             scanf("%d", &peso);
18             printf("Digite a idade: ");
19             scanf("%d", &idade);
20             if (pesado < peso)
21                 pesado = peso;
22             if (j == 1 || jovem > idade)
23                 jovem = idade;
24
25                 pesoMedio += peso;
26                 idadeMedia += idade;
27         }
28         pesoMedioTotal += pesoMedio;
29         idadeMediaTotal += idadeMedia;
30         idadeMedia = idadeMedia / (j-1);
31         pesoMedio = pesoMedio / (j-1);
32         printf("Idade media do time: %f \n", idadeMedia);
33         printf("Peso medio do time: %f \n", pesoMedio);
34         printf("Atleta mais pesado: %d \n", pesado);
35         printf("Atleta mais jovem: %d \n", jovem);
36     }
37     pesoMedioTotal = pesoMedioTotal / ((i-1) * (j-1));
38     idadeMediaTotal = idadeMediaTotal / ((i-1) * (j-1));
39     printf("Peso medio de todos os participantes: %f \n", pesoMedioTotal);
40     printf("Idade media de todos os participantes: %f \n", idadeMediaTotal)
41     ;
42 }

```

11. Construa um programa em C que leia vários números e informe quantos números entre 100 e 200 foram digitados. Quando o valor 0 (zero) for lido, o algoritmo deverá cessar sua execução.

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void main()
5  {

```

```

6     int n, cont=0;
7
8     do {
9         printf("Digite um numero: \n");
10        scanf("%d", &n);
11        if (n >= 100 && n <= 200)
12            cont++;
13    } while(n != 0);
14
15    printf("Quantidade de numeros entre 100 e 200: %d \n", cont);
16 }

```

12. Dado um país A, com 5 milhões de habitantes e uma taxa de natalidade de 3% ao ano, e um país B com 7 milhões de habitantes e uma taxa de natalidade de 2% ao ano, fazer um programa que calcule e imprima o tempo necessário para que a população do país A ultrapasse a população do país B.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int cont=0;
7     float a=5000000, b=7000000;
8
9     while(a < b) {
10        a += a * 0.03;
11        b += b * 0.02;
12        cont++;
13    }
14    printf("Total de anos: %d", cont);
15 }

```

13. Uma empresa de fornecimento de energia elétrica faz a leitura mensal dos medidores de consumo. Para cada consumidor, são digitados os seguintes dados:

- número do consumidor
- quantidade de kWh consumidos durante o mês
- tipo (código) do consumidor
  - 1-residencial, preço em reais por kWh = 0,3
  - 2-comercial, preço em reais por kWh = 0,5
  - 3-industrial, preço em reais por kWh = 0,7

Os dados devem ser lidos até que seja encontrado o consumidor com número 0 (zero). O programa deve calcular e imprimir:

- O custo total para cada consumidor
- O total de consumo para os três tipos de consumidor
- A média de consumo dos tipos 1 e 2

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int n, kwh, tipo, qtdResidencial=0, qtdComercial=0;
7     float residencial=0, comercial=0, industrial=0, total;
8     float mediaResidencial=0, mediaComercial=0;
9
10    do {
11        printf("Digite o numero do consumidor: \n");
12        scanf("%d", &n);
13        if (n == 0)
14            break;
15        printf("Digite a quantidade de kWh consumidos: \n");
16        scanf("%d", &kwh);
17        printf("TIPO \n");
18        printf("1 - Residencial \n");
19        printf("2 - Comercial \n");
20        printf("3 - Industrial \n");
21        scanf("%d", &tipo);
22        switch(tipo) {
23            case 1:
24                {
25                    total = kwh * 0.3;
26                    residencial += kwh;
27                    break;
28                }
29            case 2:
30                {
31                    total = kwh * 0.5;
32                    comercial += kwh;
33                    break;
34                }
35            case 3:
36                {
37                    total = kwh * 0.7;
38                    industrial += kwh;
39                    break;
40                }
41        }
42        if (tipo == 1)
43            qtdResidencial++;
44        else if (tipo == 2)
45            qtdComercial++;
46        printf("Custo total do consumidor: %f\n", total);
47    } while(n != 0);
48
49    if (qtdComercial != 0)
50        mediaComercial = comercial / qtdComercial;
51    if (qtdResidencial != 0)
52        mediaResidencial = residencial / qtdResidencial;
53
54    printf("Total de consumo residencial: %f \n", residencial);
```

```

55     printf("Total de consumo comercial: %f \n", comercial);
56     printf("Total de consumo industrial: %f \n", industrial);
57     printf("Media de consumo residencial: %f \n", mediaResidencial);
58     printf("Media de consumo comercial: %f \n", mediaComercial);
59 }

```

14. Faça um programa que leia vários números inteiros e apresente o fatorial de cada número. O algoritmo encerra quando se digita um número menor do que 1.

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void main()
5  {
6      int n, i, fat;
7
8      do {
9          fat = 1;
10         printf("Digite um numero: \n");
11         scanf("%d", &n);
12         if (n > 1) {
13             for (i=n; i>0; i--)
14                 fat *= i;
15             printf("\nFatorial: %d \n", fat);
16         }
17         else
18             break;
19     } while(n != 0);
20 }

```

15. Faça um programa em C que permita entrar com a idade de várias pessoas e imprima:

- total de pessoas com menos de 21 anos
- total de pessoas com mais de 50 anos

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void main()
5  {
6      int idade, pessoaMenor21=0, pessoaMaior50=0;
7
8      do {
9          printf("Digite uma idade: ");
10         scanf("%d", &idade);
11         if (idade >= 1) {
12             if (idade < 21)
13                 pessoaMenor21++;
14             else if (idade > 50)
15                 pessoaMaior50++;
16         }
17     } while (idade != 0);

```

```

18
19     printf("\nPessoas menores de 21 anos: %d", pessoaMenor21);
20     printf("\nPessoas maiores de 50 anos: %d", pessoaMaior50);
21 }

```

16. Sabendo-se que a unidade lógica e aritmética calcula a divisão por meio de subtrações sucessivas, criar um algoritmo que calcule e imprima o resto da divisão de números inteiros lidos. Para isso, basta subtrair o divisor ao dividendo, sucessivamente, até que o resultado seja menor do que o divisor. O número de subtrações realizadas corresponde ao quociente inteiro e o valor restante da subtração corresponde ao resto. Suponha que os números lidos sejam positivos e que o dividendo seja maior do que o divisor.

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void main()
5  {
6      int i, dividendo, divisor, subtracao;
7
8      printf("Digite o dividendo: ");
9      scanf("%d", &dividendo);
10     printf("Digite o divisor: ");
11     scanf("%d", &divisor);
12     if (dividendo < 0)
13         printf("Dividendo invalido \n");
14     else if (divisor < 0)
15         printf("Divisor invalido \n");
16     else if (dividendo < divisor)
17         printf("Dividendo deve ser maior que o divisor \n");
18     else {
19         subtracao = dividendo;
20         while (subtracao >= divisor) {
21             subtracao -= divisor;
22         }
23         printf("Resto da divisao: %d \n", subtracao);
24     }
25 }

```

17. Crie um programa em C que possa ler um conjunto de pedidos de compra e calcule o valor total da compra. Cada pedido é composto pelos seguintes campos:

- número de pedido
- data do pedido (dia, mês, ano)
- preço unitário
- quantidade

O programa deverá processar novos pedidos até que o usuário digite 0 (zero) como número do pedido.

```

1  #include <stdio.h>
2  #include <stdlib.h>

```

```

3
4 void main()
5 {
6     int n, quantidade, dia, mes, ano;
7     float preco, total;
8
9     do {
10        printf("Digite o numero do pedido: \n");
11        scanf("%d", &n);
12        if (n == 0)
13            break;
14        printf("Digite a data do pedido <dia/mes/ano>: \n");
15        scanf("%d %d %d", &dia, &mes, &ano);
16        printf("Digite o preco unitario: \n");
17        scanf("\n%f", &preco);
18        printf("Digite a quantidade: \n");
19        scanf("\n%d", &quantidade);
20        total= preco * quantidade;
21        printf("Custo: %.2f\n", total);
22    } while(n != 0);
23 }

```

18. Uma pousada estipulou o preço para a diária em R\$30,00 e mais uma taxa de serviços diários de:

- R\$15,00, se o número de dias for menor que 10;
- R\$8,00, se o número de dias for maior ou igual a 10;

Faça um programa que imprima o nome, a conta e o número da conta de cada cliente e ao final o total faturado pela pousada.

O programa deverá ler novos clientes até que o usuário digite 0 (zero) como número da conta.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int n, dias;
7     float total;
8     char nome[100];
9
10    do {
11        printf("Digite o numero da conta: \n");
12        scanf("%d", &n);
13        if (n == 0)
14            break;
15        printf("Digite o nome do cliente: \n");
16        fflush(stdin);
17        fgets(nome, 100, stdin);
18        printf("Digite a quantidade de dias: \n");
19        scanf("\n%d", &dias);
20
21        if (dias < 10)

```

```

22     total = (dias * 30) + (dias * 15);
23     else
24         total = (dias * 30) + (dias * 8);
25
26     printf("Custo: %.2f\n", total);
27 } while(n != 0);
28 }

```

19. Em uma Universidade, os alunos das turmas de informática fizeram uma prova de algoritmos. Cada turma possui um número de alunos. Criar um programa que imprima:

- quantidade de alunos aprovados;
- média de cada turma;
- percentual de reprovados.

**Obs.:** Considere aprovado com nota  $\geq 7.0$

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void main()
5  {
6      int i, n, aprovado=0;
7      float nota, soma=0, percentual, media, reprovados;
8
9      printf("Digite a quantidade de alunos na turma: \n");
10     scanf("%d", &n);
11     for (i=1; i<=n; i++) {
12         printf("Digite a nota do aluno: \n");
13         scanf("%f", &nota);
14         if (nota >= 7)
15             aprovado++;
16         soma += nota;
17     }
18
19     media = soma / n;
20     reprovados = n - aprovado;
21     percentual = (reprovados / n) * 100;
22
23     printf("\nMedia da turma: %f \n", media);
24     printf("\nNumero de aprovados: %d \n", aprovado);
25     printf("\nPercentual de Reprovados: %f \n", percentual);
26 }

```

20. Uma pesquisa de opinião realizada no Rio de Janeiro, teve as seguintes perguntas:

- Qual o seu time de coração?
  - 1-Fluminense;
  - 2-Botafogo;
  - 3-Vasco;
  - 4-Flamengo;
  - 5-Outros

- Onde você mora?
  - 1-RJ;
  - 2-Niterói;
  - 3-Outros
- Qual o seu salário?

Faça um programa que imprima:

- o número de torcedores por clube;
- a média salarial dos torcedores do Botafogo;
- o número de pessoas moradoras do Rio de Janeiro, torcedores de outros clubes;
- o número de pessoas de Niterói torcedoras do Fluminense

**Obs.:** O programa encerra quando se digita 0 para o time.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int time, endereco, fla=0, flu=0, vas=0, bot=0, out=0;
7     int torcOutros=0, torcFlu=0;
8     float salario, soma=0, media;
9
10    do {
11        printf("Digite o salario: \n");
12        scanf("%f", &salario);
13        printf("TIME");
14        printf("\n1 - Fluminense");
15        printf("\n2 - Vasco");
16        printf("\n3 - Botafogo");
17        printf("\n4 - Flamengo");
18        printf("\n5 - Outros\n");
19        scanf("\n%d", &time);
20        if (time == 0)
21            break;
22        else {
23            if (time == 1) {
24                flu++;
25            }
26            else if (time == 2)
27                vas++;
28            else if (time == 3) {
29                bot++;
30                soma += salario;
31            }
32            else if (time == 4)
33                fla++;
34            else {
35                out++;
36            }
37        }

```

```

38     printf("ENDERECO");
39     printf("\n1 - RJ");
40     printf("\n2 - Niteroi");
41     printf("\n3 - Outros\n");
42     scanf("\n%d", &endereco);
43
44     if (endereco == 1 && time == 4)
45         torcOutros++;
46     if (endereco == 2 && time == 1)
47         torcFlu++;
48
49     } while(time != 0);
50
51     media = soma / bot;
52     printf("Torcedores do fluminense: %d \n", flu);
53     printf("Torcedores do vasco: %d \n", vas);
54     printf("Torcedores do botafogo: %d \n", bot);
55     printf("Torcedores do flamengo: %d \n", fla);
56     printf("Torcedores de outros: %d \n", out);
57     printf("Media de salario dos torcedores do botafogo: %f \n", media);
58     printf("Do Rio que torcem para outros: %d \n", torcOutros);
59     printf("De Niteroi que torcem para o fluminense: %d \n", torcFlu);
60 }

```

21. Em uma universidade cada aluno possui os seguintes dados:

- Renda pessoal;
- Renda familiar;
- Total gasto com alimentação;
- Total gasto com outras despesas;

Faça um programa que imprima a porcentagem dos alunos que gasta acima de R\$200,00 com outras despesas. O número de alunos com renda pessoal maior que a renda familiar e a porcentagem gasta com alimentação e outras despesas em relação às rendas pessoal e familiar.

**Obs.:** O programa encerra quando se digita 0 para a renda pessoal.

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void main()
5  {
6      float rendaPessoal, rendaFamiliar, alimentacao, outros;
7      float outrasDespesas=0, total=0;
8      float percGastamOutras, percAlimentacaoP, percAlimentacaoF;
9      float percOutrasP, percOutrasF;
10     int nPessoalMaior=0;
11
12     do {
13         printf("\nDigite a renda pessoal:");
14         scanf("%f", &rendaPessoal);
15         if (rendaPessoal == 0)
16             break;

```

```

17     printf("\nDigite a renda familiar:");
18     scanf("%f", &rendaFamiliar);
19     printf("\nDigite o gasto com alimentacao: ");
20     scanf("%f", &alimentacao);
21     printf("\nDigite o gasto com outras despesas: ");
22     scanf("%f", &outros);
23     total++;
24
25     if (outros > 200)
26         outrasDespesas++;
27     if (rendaPessoal > rendaFamiliar)
28         nPessoalMaior++;
29
30     percAlimentacaoP = (alimentacao / rendaPessoal) * 100;
31     percAlimentacaoF = (alimentacao / rendaFamiliar) * 100;
32     percOutrasP = (outros / rendaPessoal) * 100;
33     percOutrasF = (outros / rendaFamiliar) * 100;
34
35     printf("\n Aliment. por renda pessoal: %f", percAlimentacaoP);
36     printf("\n Aliment. por renda familiar: %f", percAlimentacaoF);
37     printf("\n Outras por renda pessoal: %f", percOutrasP);
38     printf("\n Outras por renda familiar: %f", percOutrasF);
39 } while (rendaPessoal != 0);
40
41     percGastamOutras = (outrasDespesas / total) * 100;
42     printf("\nPerc gastam acima de 200 outras: %f ", percGastamOutras);
43     printf("\nCom renda pessoal maior familiar: %d ", nPessoalMaior);
44 }

```

22. Crie um programa que ajude o DETRAN a saber o total de recursos que foram arrecadados com a aplicação de multas de trânsito.

O algoritmo deve ler as seguintes informações para cada motorista:

- número da carteira de motorista (de 1 a 4327);
- número de multas;
- valor de cada uma das multas.

Deve ser impresso o valor da dívida para cada motorista e ao final da leitura o total de recursos arrecadados (somatório de todas as multas). O programa deverá imprimir também o número da carteira do motorista que obteve o maior número de multas.

**Obs.:** O programa encerra ao ler a carteira de motorista de valor 0.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     float carteira, valor, total=0, divida;
7     int multas, i, maiorNmultas=0, maiorCarteira;
8
9     do {
10         divida = 0;

```

```

11     i = 1;
12     printf("\nDigite o numero da carteira de motorista: ");
13     scanf("%f", &carteira);
14     if (carteira == 0)
15         break;
16     if (carteira > 4327)
17         break;
18     printf("\nDigite o numero de multas: ");
19     scanf("%d", &multas);
20     if (multas > maiorNmultas) {
21         maiorNmultas = multas;
22         maiorCarteira = carteira;
23     }
24
25     while(i <= multas) {
26         printf("\nDigite o valor da multa %d: ", i);
27         scanf("%f", &valor);
28         total += valor;
29         divida += valor;
30         i++;
31     }
32     printf("\nDivida do motorista: %f", divida);
33 } while (carteira != 0);
34
35 printf("\nTotal arrecadado pelo DETRAN: %f", total);
36 printf("\nCarteiro com maior numero de multas: %d", maiorCarteira);
37 }

```

23. Crie um programa que leia um conjunto de informações (nome, sexo, idade, peso e altura) dos atletas que participaram de uma olimpíada, e informar:

- a atleta do sexo feminino mais alta;
- o atleta do sexo masculino mais pesado;
- a média de idade dos atletas.

**Obs.:** Deverão se lidos dados dos atletas até que seja digitado o nome @ para um atleta.

**Para resolver este exercício, consulte a aula 7** que aborda o tratamento de strings, como comparação e atribuição de textos.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 void main()
6 {
7     float altura, peso, maisAlta=0, maisPesado=0, totalIdade=0;
8     int idade, sexo, total=0;
9     char nome[100], nomeAlta[100], nomePesado[100];
10
11     do {
12         printf("\nDigite o nome do atleta: ");
13         fflush(stdin);

```

```

14     gets(nome);
15     if (strcmp(nome, "@") == 0)
16         break;
17     printf("\nDigite a idade do atleta: ");
18     scanf("%d", &idade);
19     printf("\nDigite o peso do atleta: ");
20     scanf("%f", &peso);
21     printf("\nDigite a altura do atleta: ");
22     scanf("%f", &altura);
23     printf("\nDigite o sexo do atleta <1-M / 2-F>: ");
24     scanf("%d", &sexo);
25     if (sexo == 2 && altura > maisAlta) {
26         maisAlta = altura;
27         strcpy(nomeAlta, nome);
28     }
29
30     if (sexo == 1 && peso > maisPesado) {
31         maisPesado = peso;
32         strcpy(nomePesado, nome);
33     }
34     totalIdade += idade;
35     total++;
36 } while(strcmp(nome, "@") != 0);
37 printf("\nAtleta mais alta: %s\t %f", nomeAlta, maisAlta);
38 printf("\nAtleta mais pesado: %s\t %f", nomePesado, maisPesado);
39 printf("\nMedia de idade dos atletas: %f", totalIdade / total);
40 }

```

24. Faça um programa que calcule quantos litros de gasolina são usados em uma viagem, sabendo que um carro faz 10 km/litro. O usuário fornecerá a velocidade do carro e o período de tempo que viaja nesta velocidade para cada trecho do percurso. Então, usando as fórmulas **distância = tempo x velocidade** e **litros consumidos = distância / 10**, o programa computará, para todos os valores não-negativos de velocidade, os litros de combustível consumidos. O programa deverá imprimir a distância e o número de litros de combustível gastos naquele trecho. Deverá imprimir também o total de litros gastos na viagem. O programa encerra quando o usuário informar um valor negativo de velocidade.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     float velocidade, distancia, tempo, consumo;
7     float total=0;
8
9     do {
10        printf("\nDigite a velocidade: ");
11        scanf("%f", &velocidade);
12        if (velocidade < 0)
13            break;
14        printf("\nDigite o tempo: ");
15        scanf("%f", &tempo);
16        distancia = velocidade * tempo;

```

```

17     consumo = distancia / 10;
18     total += consumo;
19     printf("\nDistancia percorrida: %f ", distancia);
20     printf("\nTotal gasto no trecho: %f ", consumo);
21 } while(velocidade >= 0);
22 printf("\nTotal de litros consumidos: %f ", total);
23 }

```

25. Faça um programa que calcule o imposto de renda de um grupo de contribuintes, considerando que:

- os dados de cada contribuinte (CIC, número de dependentes e renda bruta anual) serão fornecidos pelo usuário via teclado;
- para cada contribuinte será feito um abatimento de R\$600 por dependente;
- a renda líquida é obtida diminuindo-se o abatimento com os dependentes da renda bruta anual;
- para saber quanto o contribuinte deve pagar de imposto, utiliza-se a tabela a seguir:

Renda Líquida	Imposto
até R\$1000	Isento
de R\$1001 a R\$5000	15%
acima de R\$5000	25%

- o valor de CIC igual a zero indica final de dados;
- o programa deverá imprimir, para cada contribuinte, o número do CIC e o imposto a ser pago;
- ao final o programa deverá imprimir o total do imposto arrecadado pela Receita Federal e o número de contribuintes isentos;
- leve em consideração o fato de o primeiro CIC informado poder ser zero.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     float renda, imposto=0, total=0;
7     int cic, dependentes;
8
9     do {
10        printf("\nDigite o CIC: ");
11        scanf("%d", &cic);
12        if (cic == 0)
13            break;
14        printf("\nDigite o numero de dependentes: ");
15        scanf("%d", &dependentes);
16        printf("\nDigite a renda: ");
17        scanf("%f", &renda);
18
19        renda -= dependentes * 600;

```

```

20
21     if (renda < 1000) {
22         printf("\nIsento de imposto");
23     }
24     else if (renda < 5000) {
25         imposto = renda * 0.15;
26     }
27     else {
28         imposto = renda * 0.25;
29     }
30     printf("CIC: %d \n", cic);
31     printf("Imposto: %f \n", imposto);
32     total += imposto;
33 } while(cic != 0);
34
35 printf("\nTotal arrecadado pela receita federal: %f", total);
36 }

```

26. Foi feita uma pesquisa de audiência de canal de TV em várias casas de uma certa cidade, em um determinado dia. Para cada casa visitada foram fornecidos o número do canal (4, 5, 7, 12) e o número de pessoas que estavam assistindo a ele naquela casa. Se a televisão estivesse desligada, nada seria anotado, ou seja, esta casa não entraria na pesquisa. Criar um programa que:

- Leia um número indeterminado de dados, isto é, o número do canal e o número de pessoas que estavam assistindo;
- Calcule e imprima a porcentagem de audiência em cada canal.

**Obs.:** Para encerrar a entrada de dados, digite o número do canal zero.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int opcao, cont=0;
7     float canal4=0, canal5=0, canal7=0, canal12=0;
8     float percC4, percC5, percC7, percC12;
9
10    do {
11        printf("\nMENU");
12        printf("\n1 - Canal 4");
13        printf("\n2 - Canal 5");
14        printf("\n3 - Canal 7");
15        printf("\n4 - Canal 12");
16        printf("\nDigite a opcao: ");
17        scanf("%d", &opcao);
18        if (opcao == 0)
19            break;
20        cont++;
21        switch (opcao) {
22            case 1:
23                canal4++;
24                break;

```

```

25     case 2:
26         canal5++;
27         break;
28     case 3:
29         canal7++;
30         break;
31     case 4:
32         canal12++;
33         break;
34     default:
35         printf("Canal invalido!\n");
36     }
37 } while(opcao != 0);
38
39 if (cont == 0)
40     cont = 1;
41 percC4 = (canal4 / cont) * 100;
42 percC5 = (canal5 / cont) * 100;
43 percC7 = (canal7 / cont) * 100;
44 percC12 = (canal12 / cont) * 100;
45 printf("\nTotal de pessoas entrevistadas: %d", cont);
46 printf("\nPerc de pessoas que assistiam ao canal 4: %f", percC4);
47 printf("\nPerc de pessoas que assistiam ao canal 5: %f", percC5);
48 printf("\nPerc de pessoas que assistiam ao canal 7: %f", percC7);
49 printf("\nPerc de pessoas que assistiam ao canal 12: %f", percC12);
50 }

```

27. Crie um programa que calcule e imprima o CR do período para os alunos de computação. Para cada aluno, o algoritmo deverá ler:

- número da matrícula;
- quantidade de disciplinas cursadas;
- notas em cada disciplina;  
Além do CR de cada aluno, o programa deve imprimir o melhor CR dos alunos que cursaram 5 ou mais disciplinas.
- fim da entrada de dados é marcada por uma matrícula inválida (matrículas válidas de 1 a 5000);
- CR do aluno é igual à média aritmética de suas notas.

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void main()
5  {
6      int matricula, disciplinas, i;
7      float cr, maior=0, nota, notas;
8
9      do {
10         notas = 0;
11         printf("\nDigite a matricula: ");
12         scanf("%d", &matricula);
13         if (matricula < 1 || matricula > 5000)

```

```

14         break;
15     printf("\nDigite a quantidade de disciplinas cursadas: ");
16     scanf("%d", &disciplinas);
17     i = 1;
18     while(i <= disciplinas) {
19         printf("\nDigite a nota da disciplina %d: ", i);
20         scanf("%f", &nota);
21         notas += nota;
22         i++;
23     }
24     cr = notas / disciplinas;
25     printf("\nMatricula: %d", matricula);
26     printf("\nNumero de disciplinas: %d", disciplinas);
27     printf("\nCR: %f", cr);
28
29     if (disciplinas > 4 && cr > maior) {
30         maior = cr;
31     }
32 } while(matricula > 0 && matricula <= 5000);
33
34 printf("\nMaior CR dos alunos: %f", maior);
35 }

```

28. Construa um programa que receba a idade, a altura e o peso de várias pessoas, Calcule e imprima:

- a quantidade de pessoas com idade superior a 50 anos;
- a média das alturas das pessoas com idade entre 10 e 20 anos;
- a porcentagem de pessoas com peso inferior a 40 quilos entre todas as pessoas analisadas.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int idade, total=0, mais50=0, cont=0;
7     float peso, altura, somaAltura=0, mediaAltura;
8     float menos40=0, percPeso40;
9
10    do {
11        printf("\nDigite a idade ou zero para encerrar: ");
12        scanf("%d", &idade);
13        if (idade == 0)
14            break;
15        printf("\nDigite a altura: ");
16        scanf("%f", &altura);
17        printf("\nDigite o peso: ");
18        scanf("%f", &peso);
19        total++;
20        if (idade > 50)
21            mais50++;
22    }

```

```

23     if (idade > 9 && idade < 21) {
24         somaAltura += altura;
25         cont++;
26     }
27
28     if (peso < 40)
29         menos40++;
30 } while(idade > 0);
31 if (total == 0)
32     total = 1;
33 if (cont == 0)
34     cont = 1;
35 mediaAltura = somaAltura / cont;
36 percPeso40 = menos40 / total;
37 printf("\nQuantidade de pessoas com mais de 50 anos: %d", mais50);
38 printf("\nMedia de altura entre 10 e 20 anos: %f", mediaAltura);
39 printf("\nPorcentagem com peso inferior a 40 quilos: %f", percPeso40);
40 }

```

29. Construa um programa que receba o valor e o código de várias mercadorias vendidas em um determinado dia. Os códigos obedecem a lista a seguir:

L-limpeza

A-Alimentação

H-Higiene

**Calcule e imprima:**

- o total vendido naquele dia, com todos os códigos juntos;
- o total vendido naquele dia em cada um dos códigos.

**Obs.:** Para encerrar a entrada de dados, digite o valor da mercadoria zero.

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void main()
5  {
6      float valor, limpeza=0, alimentacao=0, higiene=0, total=0;
7      char codigo;
8
9      do {
10         printf("\nDigite a valor: ");
11         scanf("%f", &valor);
12         if (valor == 0)
13             break;
14         printf("\nDigite o codigo: ");
15         fflush(stdin);
16         scanf("%c", &codigo);
17         if (codigo == 'l' || codigo == 'L')
18             limpeza += valor;
19         else if (codigo == 'a' || codigo == 'A')
20             alimentacao += valor;
21         else
22             higiene += valor;
23         total += valor;

```

```

24     } while(valor > 0);
25
26     printf("\nTotal: %f",total);
27     printf("\nTotal de itens de limpeza: %f ",limpeza);
28     printf("\nTotal de itens de alimentacao: %f ",alimentacao);
29     printf("\nTotal de itens de higiene: %f ",higiene);
30 }

```

30. Faça um programa que receba a idade e o estado civil (C-casado, S-solteiro, V-viúvo e D-desquitado ou separado) de várias pessoas. Calcule e imprima:

- a quantidade de pessoas casadas;
- a quantidade de pessoas solteiras;
- a média das idades das pessoas viúvas;
- a porcentagem de pessoas desquitadas ou separadas dentre todas as pessoas analisadas.

**Obs.:** Para encerrar a entrada de dados, digite um número menor que zero para a idade.

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void main()
5  {
6      int idade, solteiro=0, casado=0, total=0, viuvo=0;
7      char codigo;
8      float idadeViuvo=0, mediaIdadeV, outros=0, percOutros;
9
10     do {
11         printf("\nDigite a idade: ");
12         scanf("%d", &idade);
13         if (idade < 0)
14             break;
15         total++;
16         printf("\nC - casado");
17         printf("\nS - solteiro");
18         printf("\nV - viuvo");
19         printf("\nD - separado");
20         printf("\nDigite o estado civil: ");
21         fflush(stdin);
22         scanf("%c", &codigo);
23         if (codigo == 'c' || codigo == 'C')
24             casado++;
25         else if (codigo == 's' || codigo == 'S')
26             solteiro++;
27         else if (codigo == 'v' || codigo == 'V') {
28             idadeViuvo += idade;
29             viuvo++;
30         }
31         else
32             outros++;
33     } while(idade >= 0);

```

```
34
35     if (total == 0)
36         total = 1;
37     mediaIdadeV = idadeViuvo / viuvo;
38     percOutros = outros / total;
39     printf("\nQuantidade de casados: %d", casado);
40     printf("\nQuantidade de solteiros: %d", solteiro);
41     printf("\nMedia de idade dos viuvos: %f", mediaIdadeV);
42     printf("\nPorcentagem de pessoas separadas: %f", percOutros);
43 }
```

## Exercícios Resolvidos da Aula 4

1. Faça um programa em C que armazene 15 números inteiros em um vetor e depois permita que o usuário digite um número inteiro para ser buscado no vetor, se for encontrado o programa deve imprimir a posição desse número no vetor, caso contrário, deve imprimir a mensagem: "Nao encontrado!".

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int v[15], n, aux=0, i;
7
8     printf("\nDigite os 15 valores do vetor:\n");
9     for (i=0; i<15; i++)
10         scanf("%d", &v[i]);
11
12     printf("\nDigite um numero para procurar no vetor:\n");
13     scanf("%d", &n);
14     for (i=0; i<15; i++) {
15         if (v[i] == n) {
16             printf("\nNumero encontrado na posicao %d\n", i+1);
17             aux = 1;
18             break;
19         }
20     }
21
22     if (aux == 0)
23         printf("\nNumero nao encontrado\n");
24 }
```

2. Faça um programa que armazene 10 letras em um vetor e imprima uma listagem numerada.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
```

```
6   int i;
7   char letras[10];
8
9   printf("\nDigite 10 letras:\n");
10  for (i=0; i<10; i++) {
11      fflush(stdin);
12      scanf("%c", &letras[i]);
13  }
14
15  printf("\nLista com as letras numeradas\n");
16  for (i=0; i<10; i++)
17      printf("%d Letra: %c \n", i+1, letras[i]);
18 }
```

3. Construa um programa em C que armazene 15 números em um vetor e imprima uma listagem numerada contendo o número e uma das mensagens: par ou ímpar.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void main()
5  {
6      int v[15], i;
7
8      printf("\nDigite os 15 valores do vetor:\n");
9      for (i=0; i<15; i++)
10         scanf("%d", &v[i]);
11
12     printf("Posicao \t Numero \n");
13     for (i=0; i<15; i++) {
14         if ((v[i] % 2) == 0)
15             printf("%d \t\t %d \t par\n", i, v[i]);
16         else
17             printf("%d \t\t %d \t impar\n", i, v[i]);
18     }
19 }
```

4. Faça um programa que armazene 8 números em um vetor e imprima todos os números. Ao final, imprima o total de números múltiplos de seis.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void main()
5  {
6      int v[8], i, cont=0;
7
8      printf("\nDigite os 8 valores do vetor:\n");
9      for (i=0; i<8; i++)
10         scanf("%d", &v[i]);
11
12     for (i=0; i<8; i++)
13         if ((v[i] % 6) == 0)
14             cont++;
```

```

15
16     for (i=0; i<8; i++)
17         printf("\nNumero: %d\n", v[i]);
18     printf("\nMultiplos de 6: %d\n", cont);
19 }

```

5. Faça um programa que armazene as notas das provas 1 e 2 de 15 alunos. Calcule e armazene a média arredondada. Armazene também a situação do aluno: 1-Aprovado ou 2-Reprovado. Ao final o programa deve imprimir uma listagem contendo as notas, a média e a situação de cada aluno em formato tabulado. Utilize quantos vetores forem necessários para armazenar os dados.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int i, situacoes[15];
7     float notasp1[15], notasp2[15], medias[15];
8
9     for (i=0; i<15; i++) {
10        printf("\nDigite a nota 1 do aluno: \n");
11        scanf("%f", &notasp1[i]);
12        printf("\nDigite a nota 2 do aluno: \n");
13        scanf("%f", &notasp2[i]);
14        medias[i] = (notasp1[i] + notasp2[i]) / 2;
15        if (medias[i] >= 7) {
16            situacoes[i] = 1;
17        }
18        else {
19            situacoes[i] = 2;
20        }
21    }
22    printf("\nLista dos alunos\n");
23    printf("Nota 1 \t\t Nota 2 \t Media \t\t Situacao \n");
24    for (i=0; i<15; i++) {
25        printf("%f\t", notasp1[i]);
26        printf("%f\t", notasp2[i]);
27        printf("%f\t", medias[i]);
28        if (situacoes[i] == 1)
29            printf("Aprovado \n");
30        else
31            printf("Reprovado \n");
32    }
33 }

```

6. Construa um programa que permita armazenar o salário de 20 pessoas. Calcular e armazenar o novo salário sabendo-se que o reajuste foi de 8%. Imprimir uma listagem numerada com o salário e o novo salário. Declare quantos vetores forem necessários.

```

1 #include <stdio.h>
2 #include <stdlib.h>

```

```

3
4 void main()
5 {
6     int i;
7     float salarios[20], salariosReaj[20];
8
9     for (i=0; i<20; i++) {
10        printf("\nDigite o salario: \n");
11        scanf("%f",&salarios[i]);
12        salariosReaj[i] = salarios[i] + (salarios[i] * 0.08);
13    }
14
15    printf("\nNumero \tSalario \t Salario Reajustado\n");
16    for (i=0; i<20; i++) {
17        printf("%d \t",i+1);
18        printf("%f \t",salarios[i]);
19        printf("%f \n",salariosReaj[i]);
20    }
21 }

```

7. Crie um programa que leia o preço de compra e o preço de venda de 100 mercadorias (utilize vetores). Ao final, o programa deverá imprimir quantas mercadorias proporcionam:

- lucro < 10%
- 10% <= lucro <= 20%
- lucro > 20%

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int i, menos10=0, menos20=0, mais20=0;
7     float precoCompra[100], precoVenda[100], lucro;
8
9     for (i=0; i<100; i++) {
10        printf("\nDigite o preco de compra do produto:\n");
11        scanf("%f", &precoCompra[i]);
12        printf("\nDigite o preco de venda do produto:\n");
13        scanf("%f", &precoVenda[i]);
14        lucro = (precoVenda[i] / precoCompra[i]) * 100;
15        if (lucro < 10)
16            menos10++;
17        else if (lucro <= 20)
18            menos20++;
19        else
20            mais20++;
21    }
22    printf("\nQtde com ate 10 por cento de lucro: %d\n", menos10);
23    printf("\nQtde com ate 20 por cento de lucro: %d\n", menos20);
24    printf("\nQtde com mais de 20 por cento de lucro: %d\n", mais20);
25 }

```

8. Construa um programa que armazene o código, a quantidade, o valor de compra e o valor de venda de 30 produtos. A listagem pode ser de todos os produtos ou somente de um ao se digitar o código.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int i, codigo[3], quantidade[3], resposta;
7     float vlrcompra[3], vlrventa[3];
8
9     for (i=0; i<3; i++) {
10        printf("\nDigite o codigo do produto:\n");
11        scanf("%d", &codigo[i]);
12        printf("\nDigite a quantidade do produto:\n");
13        scanf("%d", &quantidade[i]);
14        printf("\nDigite o valor de compra do produto:\n");
15        scanf("%f", &vlrcompra[i]);
16        printf("\nDigite o valor de venda do produto:\n");
17        scanf("%f", &vlrventa[i]);
18    }
19
20    printf("\n1 - listar todos os produtos\n");
21    printf("\n2 - listar produto por codigo\n");
22    scanf("%d", &resposta);
23    if (resposta == 1) {
24        printf("\nCodigo \t Quantidade \t Valor Compra \t Valor Venda \n");
25        for (i=0; i<3; i++) {
26            printf("%d \t",codigo[i]);
27            printf("%d \t\t",quantidade[i]);
28            printf("%f \t",vlrcompra[i]);
29            printf("%f \n",vlrventa[i]);
30        }
31    }
32    else if (resposta == 2) {
33        printf("\nDigite o codigo do produto:\n");
34        scanf("%d", &resposta);
35        printf("\nCodigo \t Quantidade \t Valor Compra \t Valor Venda \n");
36        for (i=0; i<3; i++) {
37            if (codigo[i] == resposta) {
38                printf("%d \t",codigo[i]);
39                printf("%d \t\t",quantidade[i]);
40                printf("%f \t",vlrcompra[i]);
41                printf("%f \n",vlrventa[i]);
42                break;
43            }
44        }
45    }
46    else
47        printf("Opcao invalida! \n");
48 }
```

9. Faça um programa em C que leia dois conjuntos de números inteiros, tendo

cada um 10 elementos. Ao final o programa deve listar os elementos comuns aos conjuntos.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int v1[10], v2[10], v3[10], i, j, cont=0;
7
8     printf("\nDigite os 10 numeros do vetor 1:\n");
9     for (i=0; i<10; i++)
10         scanf("%d", &v1[i]);
11
12     printf("\nDigite os 10 numeros do vetor 2:\n");
13     for (i=0; i<10; i++)
14         scanf("%d", &v2[i]);
15
16     for (i=0; i<10; i++) {
17         for (j=0; j<10; j++) {
18             if (v1[i] == v2[j]) {
19                 v3[cont] = v1[i];
20                 cont++;
21                 break;
22             }
23         }
24     }
25
26     printf("\nElementos comuns aos conjuntos:\n");
27     for (i=0; i<cont; i++) {
28         printf("%d \n", v3[i]);
29     }
30 }

```

10. Faça um programa que leia um vetor **vet** de 10 elementos e obtenha um vetor **w** cujos componentes são os fatoriais dos respectivos componentes de **vet**.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int vet[10], w[10], i, j, aux=0;
7
8     printf("\nDigite os 10 valores do vetor:\n");
9     for (i=0; i<10; i++)
10         scanf("%d", &vet[i]);
11
12     for (i=0; i<10; i++) {
13         aux=1;
14         if (vet[i] == 1)
15             w[i] = 1;
16         else {
17             for (j=1; j<vet[i]; j++) {
18                 aux *= j + 1;

```

```

19     }
20     w[i] = aux;
21 }
22 }
23 printf("\nNumero \t Fatorial:\n");
24 for (i=0; i<10; i++)
25     printf("%d \t %d \n", vet[i], w[i]);
26 }

```

11. Construa um programa que leia dados para um vetor de 100 elementos inteiros. Imprimir o maior e o menor, sem ordenar, o percentual de números pares e a média dos elementos do vetor.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int v[100], i, maior, menor;
7     float media=0, pares=0, perPares;
8
9     printf("\nDigite os 100 valores do vetor:\n");
10    for (i=0; i<100; i++)
11        scanf("%d", &v[i]);
12
13    maior = v[0];
14    menor = v[0];
15    for (i=1; i<100; i++) {
16        if (v[i] > maior)
17            maior = v[i];
18        if (v[i] < menor)
19            menor = v[i];
20        media += v[i];
21        if ((v[i] % 2) == 0)
22            pares++;
23    }
24    perPares = (pares / 100);
25    media = media / 100;
26    printf("\nMaior elemento do vetor: %d\n", maior);
27    printf("\nMenor elemento do vetor: %d\n", menor);
28    printf("\nMedia dos elemento do vetor: %.2f \n", media);
29    printf("\nPorcentagem de numeros pares: %.2f \n", perPares);
30 }

```

12. Crie um programa para gerenciar um sistema de reservas de mesas em uma casa de espetáculo. A casa possui 30 mesas de 5 lugares cada. O programa deverá permitir que o usuário escolha o código de uma mesa (100 a 129) e forneça a quantidade de lugares desejados. O programa deverá informar se foi possível realizar a reserva e atualizar a reserva. Se não for possível, o programa deverá emitir uma mensagem. O programa deve terminar quando o usuário digitar o código **0 (zero)** para uma mesa ou quando todos os 150 lugares estiverem ocupados.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int contCadeiras=0, numero, cadeiras, i;
7     int mesas[30];
8
9     for (i=0; i<30; i++)
10         mesas[i] = 0;
11     do {
12         printf("\nDigite o numero da mesa para reserva:\n");
13         scanf("%d", &numero);
14         if (numero > 100 && numero < 130) {
15             printf("\nDigite o numero de cadeiras\n");
16             scanf("%d", &cadeiras);
17             if (cadeiras > 0 && cadeiras < 6) {
18                 int n = numero % 100;
19                 if (mesas[n] + cadeiras < 6) {
20                     printf("\nFoi possivel reservar/atualizar a reserva!");
21                     mesas[n] += cadeiras;
22                     contCadeiras += cadeiras;
23                 }
24                 else
25                     printf("\nMesa nao possui a quantidade de cadeiras
26                             solicitada\n");
27             }
28             else
29                 printf("\nMesa nao possui a quantidade de cadeiras
30                         solicitada\n");
31         }
32         else
33             printf("\nNumero invalido\n");
34         if (contCadeiras == 150)
35             break;
36     } while(numero != 0);
37 }

```

13. Construa um programa que realize as reservas de passagens aéreas de uma companhia. O programa deve permitir cadastrar o número de 10 voos e definir a quantidade de lugares disponíveis para cada um. Após o cadastro, leia vários pedidos de reserva, constituídos do número da carteira de identidade do cliente e do número do voo desejado. Para cada cliente, verificar se há possibilidade no voo desejado. Em caso afirmativo, imprimir o número da identidade do cliente e o número do voo, atualizando o número de lugares disponíveis. Caso contrário, avisar ao cliente a inexistência de lugares. A leitura do número 0 (zero) para o voo desejado indica o término da leitura de reservas.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {

```

```

6     int numero, id, i, cont;
7     int num[10], lugares[10], acento[10];
8
9     for (i=0; i<10; i++) {
10        printf("\nDigite o numero do voo:\n");
11        scanf("%d",&num[i]);
12        printf("\nDigite a quantidade de lugares no voo:\n");
13        scanf("%d",&lugares[i]);
14        acento[i] = 0;
15    }
16
17    do {
18        printf("\nDigite o numero do voo para o cliente:\n");
19        scanf("%d", &numero);
20        cont = 0;
21        for(i=0; i<10; i++) {
22            if(numero == num[i]) {
23                if (acento[i] < lugares[i]) {
24                    printf("\nDigite a identidade do cliente:\n");
25                    scanf("%d", &id);
26                    printf("\nVoo: %d\n", num[i]);
27                    printf("\nID: %d\n", id);
28                    acento[i]++;
29                }
30                else
31                    printf("\nVoo cheio\n");
32            }
33            else
34                cont++;
35        }
36        if (cont == 10)
37            printf("\nVoo nao encontrado\n");
38    } while(numero != 0);
39 }

```

14. Faça um programa que armazene 50 números inteiros em um vetor. O programa deve gerar e imprimir um segundo vetor em que cada elemento é o quadrado do elemento do primeiro vetor.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int v[50], v2[50], i;
7
8     printf("\nDigite os 50 valores do vetor:\n");
9     for (i=0; i<50; i++) {
10        scanf("%d", &v[i]);
11        v2[i] = v[i] * v[i];
12    }
13
14    printf("\nQuadrado dos elementos do vetor:\n");
15    for (i=0; i<50; i++)

```

```

16     printf("%d \n", v2[i]);
17 }

```

15. Faça um programa que leia e armazene vários números, até digitar o número 0. Imprimir quantos números iguais ao último número foram lidos. O limite de números é 100.

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void main()
5  {
6      int v[100], i, numero, cont=0, ultimo, aux=0;
7
8      printf("\nDigite os numeros:\n");
9      for (i=0; i<100; i++) {
10         scanf("%d", &v[i]);
11         if (v[i] == 0)
12             break;
13         else {
14             cont++;
15             ultimo = v[i];
16         }
17     }
18
19     for (i=0; i<cont; i++) {
20         if (v[i] == ultimo)
21             aux++;
22     }
23     printf("\nQuantidade de numeros iguais ao ultimo: %d\n", aux);
24 }

```

16. Crie um programa em C para ler um conjunto de 100 números reais e informe:

- quantos números lidos são iguais a 30
- quantos são maior que a média
- quantos são iguais a média

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void main()
5  {
6      float v[100], media=0;
7      int igual30=0, maior=0, igual=0, i;
8
9      printf("\nDigite os numeros:\n");
10     for (i=0; i<100; i++) {
11         scanf("%f",&v[i]);
12         if (v[i] == 30)
13             igual30++;
14         media += v[i];
15     }

```

```

16
17     media = media / 100;
18     for (i=0; i<100; i++) {
19         if (v[i] > media)
20             maior++;
21         else if (v[i] == media)
22             igual++;
23     }
24     printf("\nQtde de numeros iguais a 30: %d\n", igual30);
25     printf("\nQtde de numeros maiores que a media: %d\n", maior);
26     printf("\nQtde de numeros iguais a media: %d\n", igual);
27 }

```

17. Faça um programa que leia um conjunto de 30 valores inteiros, armazene-os em um vetor e os imprima ao contrário da ordem de leitura.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int v[30], i;
7
8     printf("\nDigite os numeros:\n");
9     for (i=0; i<30; i++)
10         scanf("%d", &v[i]);
11     printf("\nNumeros em ordem contraria a leitura:\n");
12     for (i=29; i>=0; i--)
13         printf("%d \n", v[i]);
14 }

```

18. Faça um programa em C que permita entrar com dados para um vetor VET do tipo inteiro com 20 posições, em que podem existir vários elementos repetidos. Gere um vetor VET2 ordenado a partir do vetor VET e que terá apenas os elementos não repetidos.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int vet[20], vet2[20], i, j, aux, cont=0;
7
8     //lendo os dados
9     printf("\nDigite os numeros:\n");
10    for (i=0; i<20; i++)
11        scanf("%d", &vet[i]);
12
13    //ordenando os dados
14    for (i=0; i<20-1; i++) {
15        for (j=i+1; j<20; j++) {
16            if (vet[i] > vet[j]) {
17                aux = vet[i];
18                vet[i] = vet[j];

```

```

19         vet[j] = aux;
20     }
21 }
22 }
23 for (i=0; i<20; i++) {
24     if (i == 0 | vet[i-1] != vet[i]) {
25         vet2[cont] = vet[i];
26         cont++;
27     }
28 }
29 //imprimindo os dados
30 printf("\nVetor ordenado e sem elementos repetidos\n");
31 for (i=0; i<cont; i++)
32     printf("%d \n", vet2[i]);
33 }

```

19. Suponha dois vetores de 30 elementos cada, contendo: código e telefone. Faça um programa que permita buscar pelo código e imprimir o telefone.

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void main()
5  {
6      int codigo[30], tel[30], i, cod;
7
8      for (i=0; i<30; i++) {
9          printf("\nDigite o codigo:\n");
10         scanf("%d", &codigo[i]);
11         printf("\nDigite o telefone:\n");
12         scanf("%d", &tel[i]);
13     }
14     printf("\nDigite o codigo para procurar:\n");
15     scanf("%d", &cod);
16     for (i=0; i<30; i++) {
17         if (codigo[i] == cod) {
18             printf("\nTelefone: %d\n", tel[i]);
19         }
20     }
21 }

```

20. Faça um programa que leia a matrícula e a média de 100 alunos. Ordene da maior para a menor nota e imprima uma relação contendo todas as matrículas e médias.

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void main()
5  {
6      int matricula[100], i, j, auxMat;
7      float media[100], auxMedia;
8
9      //lendo os dados
10     for (i=0; i<100; i++) {

```

```
11     printf("\nDigite a matricula:\n");
12     scanf("%d", &matricula[i]);
13     printf("\nDigite a media:\n");
14     scanf("%f", &media[i]);
15 }
16 //ordenando os dados
17 for (i=0; i<100-1; i++) {
18     for (j=i+1; j<100; j++) {
19         if (media[i] < media[j]) {
20             auxMedia = media[i];
21             media[i] = media[j];
22             media[j] = auxMedia;
23             auxMat = matricula[i];
24             matricula[i] = matricula[j];
25             matricula[j] = auxMat;
26         }
27     }
28 }
29 //imprimindo os dados
30 printf("\nMatricula \t Media\n");
31 for (i=0; i<100; i++) {
32     printf("%d\t\t", matricula[i]);
33     printf("%f\n", media[i]);
34 }
35 }
```



## Exercícios Resolvidos da Aula 5

1. Faça um programa em C que leia os elementos de uma matriz do tipo inteiro com tamanho 10 X 10. Ao final, imprima todos os elementos.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int m[10][10], i, j;
7
8     printf("\nDigite os 100 valores da matriz:\n");
9     for (i=0; i<10; i++)
10         for (j=0; j<10; j++)
11             scanf("%d", &m[i][j]);
12
13     printf("\nElementos da matriz:\n");
14     for (i=0; i<10; i++) {
15         printf("\n");
16         for (j=0; j<10; j++)
17             printf("%d \t", m[i][j]);
18     }
19 }
```

2. Faça um programa que leia os elementos de uma matriz do tipo inteiro com tamanho 3 X 3 e imprima os elementos multiplicando por 2.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int m[3][3], i, j;
7
8     printf("\nDigite os 9 valores da matriz:\n");
9     for (i=0; i<3; i++) {
10         for (j=0; j<3; j++) {
11             scanf("%d", &m[i][j]);
12             m[i][j] = m[i][j] * 2;
13         }
14     }
```

```

13     }
14     }
15     printf("\nElementos da matriz multiplicados:\n");
16     for (i=0; i<3; i++) {
17         printf("\n");
18         for (j=0; j<3; j++)
19             printf("%d \t", m[i][j]);
20     }
21 }

```

3. Crie um programa que armazene dados inteiros em uma matriz de ordem 5 e imprima: Todos os elementos que se encontram em posições cuja linha mais coluna formam um número par.

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void main()
5  {
6      int m[5][5], i, j;
7
8      printf("\nDigite os 25 valores da matriz:\n");
9      for (i=0; i<5; i++) {
10         for (j=0; j<5; j++) {
11             scanf("%d", &m[i][j]);
12         }
13     }
14     printf("\nElementos cuja linha + coluna = par:\n");
15     for (i=0; i<5; i++) {
16         for (j=0; j<5; j++) {
17             if (((j+i) % 2) == 0)
18                 printf("%d \n", m[i][j]);
19         }
20     }
21 }

```

4. Construa um programa que armazene dados em uma matriz de ordem 4 e imprima: Todos os elementos com números ímpares.

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void main()
5  {
6      int m[4][4], i, j;
7
8      printf("\nDigite os 20 valores da matriz:\n");
9      for (i=0; i<4; i++) {
10         for (j=0; j<4; j++) {
11             scanf("%d", &m[i][j]);
12         }
13     }
14     printf("\nElementos impares da matriz:\n");
15     for (i=0; i<4; i++) {

```

```

16     for (j=0; j<4; j++) {
17         if ((m[i][j] % 2) != 0)
18             printf("%d \n", m[i][j]);
19     }
20 }
21 }

```

5. Faça um programa que permita entrar com valores em uma matriz **A** de tamanho 3 X 4. Gerar e imprimir uma matriz **B** que é o triplo da matriz **A**.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int A[3][4], B[3][4], i, j;
7
8     printf("\nDigite os 12 valores da matriz:\n");
9     for (i=0; i<3; i++) {
10         for (j=0; j<4; j++) {
11             scanf("%d", &A[i][j]);
12             B[i][j] = A[i][j] * 3;
13         }
14     }
15     printf("\nElementos da matriz:\n");
16     for (i=0; i<3; i++) {
17         printf("\n");
18         for (j=0; j<4; j++) {
19             printf("%d \t", B[i][j]);
20         }
21     }
22 }

```

6. Crie um programa que leia valores inteiros em uma matriz **A**[2][2] e em uma matriz **B**[2][2]. Gerar e imprimir a matriz **SOMA**[2][2].

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int A[2][2], B[2][2], SOMA[2][2], i, j;
7
8     printf("\nDigite os 4 valores da matriz A:\n");
9     for (i=0; i<2; i++)
10         for (j=0; j<2; j++)
11             scanf("%d", &A[i][j]);
12
13     printf("\nDigite os 4 valores da matriz B:\n");
14     for (i=0; i<2; i++)
15         for (j=0; j<2; j++)
16             scanf("%d", &B[i][j]);
17
18     printf("\nSoma das matrizes:\n");

```

```

19     for (i=0; i<2; i++)
20         for (j=0; j<2; j++)
21             SOMA[i][j] = A[i][j] + B[i][j];
22
23     for (i=0;i<2;i++) {
24         printf("\n");
25         for (j=0; j<2; j++)
26             printf("%d \t", SOMA[i][j]);
27     }
28 }

```

7. Construa um programa para ler valores para duas matrizes do tipo inteiro de ordem 3. Gerar e imprimir a matriz diferença.

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void main()
5  {
6      int A[3][3], B[3][3], diferenca[3][3], i, j;
7
8      printf("\nDigite os 9 valores da primeira matriz:\n");
9      for (i=0; i<3; i++)
10         for (j=0; j<3; j++)
11             scanf("%d", &A[i][j]);
12
13     printf("\nDigite os 9 valores da segunda matriz:\n");
14     for (i=0; i<3; i++)
15         for (j=0; j<3; j++)
16             scanf("%d", &B[i][j]);
17
18     printf("\nMatriz diferenca:\n");
19     for (i=0; i<3; i++)
20         for (j=0; j<3; j++)
21             diferenca[i][j] = A[i][j] - B[i][j];
22
23     for (i=0; i<3; i++) {
24         printf("\n");
25         for (j=0; j<3; j++)
26             printf("%d \t", diferenca[i][j]);
27     }
28 }

```

8. Faça um programa que leia uma matriz 4 X 5 de inteiros, calcule e imprima a soma de todos os seus elementos.

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void main()
5  {
6      int m[4][5], i, j, soma=0;
7
8      printf("\nDigite os 20 valores da matriz:\n");

```

```

9     for (i=0; i<4; i++) {
10         for (j=0; j<5; j++) {
11             scanf("%d", &m[i][j]);
12             soma += m[i][j];
13         }
14     }
15     printf("\nSoma dos elementos da matriz:\n");
16     printf("%d", soma);
17 }

```

9. Construa um programa em C que leia valores inteiros para a matriz  $A_{3 \times 5}$ . Gerar e imprimir a matriz SOMALINHA, em que cada elemento é a soma dos elementos de uma linha da matriz A. Faça o trecho que gera a matriz separado da entrada e da saída.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int matriz[3][5], somalinha[3], i, j;
7
8     for (i=0; i<3; i++)
9         somalinha[i] = 0;
10    printf("\nDigite os 15 valores da matriz:\n");
11    for (i=0; i<3; i++) {
12        for (j=0; j<5; j++) {
13            scanf("%d", &matriz[i][j]);
14        }
15    }
16
17    for (i=0; i<3; i++) {
18        for (j=0; j<5; j++) {
19            somalinha[i] += matriz[i][j];
20        }
21    }
22
23    printf("\nSoma das linhas da matriz:\n");
24    for (i=0; i<3; i++)
25        printf("%d \t", somalinha[i]);
26 }

```

10. Construa um programa em C que leia valores inteiros para a matriz  $A_{3 \times 5}$ . Gerar e imprimir a matriz SOMACOLUNA, em que cada elemento é a soma dos elementos de uma coluna da matriz A. Faça o trecho que gera a matriz separado da entrada e da saída.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int matriz[3][5], somacoluna[5], i, j;
7

```

```

8   for (i=0; i<5; i++)
9       somacoluna[i] = 0;
10  printf("\nDigite os 15 valores da matriz:\n");
11  for (i=0; i<3; i++) {
12      for (j=0; j<5; j++)    {
13          scanf("%d", &matriz[i][j]);
14      }
15  }
16  for (i=0; i<5; i++) {
17      for (j=0; j<3; j++)    {
18          somacoluna[i] += matriz[j][i];
19      }
20  }
21
22  printf("\nSoma das colunas da matriz:\n");
23  for (i=0; i<5; i++)
24      printf("%d \t", somacoluna[i]);
25 }

```

11. Entrar com valores para uma matriz  $C_{2 \times 3}$ . Gerar e imprimir a  $C^t$ . A matriz transposta é gerada trocando linha por coluna. Veja o exemplo a seguir:

$$C = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \quad C^t = \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix}$$

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void main()
5  {
6      int c[2][3], transposta[3][2], i, j;
7
8      printf("\nDigite os 6 valores da matriz:\n");
9      for (i=0; i<2; i++) {
10         for (j=0; j<3; j++)
11             scanf("%d", &c[i][j]);
12     }
13
14     for (i=0; i<2; i++) {
15         for (j=0; j<3; j++)
16             transposta[j][i] = c[i][j];
17     }
18
19     printf("\nElementos da matriz transposta:\n");
20     for (i=0; i<3; i++) {
21         printf("\n");
22         for (j=0; j<2; j++)
23             printf("%d \t", transposta[i][j]);
24     }
25 }

```

12. Uma floricultura conhecedora de sua clientela gostaria de fazer um programa que pudesse controlar sempre um estoque mínimo de determinadas plantas, pois todo dia, pela manhã, o dono faz novas aquisições. Criar um algoritmo que deixe cadastrar 50 plantas, nome, estoque mínimo, estoque atual. Imprimir ao final do programa uma lista das plantas que devem ser adquiridas.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     char nomes[50][100];
7     //coluna 0 - estoque atual / coluna 1 - estoque minimo
8     int estoque[50][2];
9     int i;
10
11     printf("\nDigite os dados das 50 plantas:\n");
12     for (i=0; i<5; i++) {
13         printf("\nDigite o nome da planta:\n");
14         fflush(stdin);
15         fgets(nomes[i], 100, stdin);
16         printf("\nDigite o estoque atual:\n");
17         scanf("%d", &estoque[i][0]);
18         printf("\nDigite o estoque minimo:\n");
19         scanf("%d", &estoque[i][1]);
20     }
21     printf("Plantas com estoque baixo\n");
22     printf("Nome \t\t\t Estoque Atual \t Estoque Minimo\n");
23     for (i=0; i<5; i++) {
24         if (estoque[i][0] < estoque[i][1])
25             printf("%s \t\t\t ", nomes[i]);
26             printf("%d \t\t %d \n", estoque[i][0], estoque[i][1]);
27     }
28 }

```

13. A gerente do cabeleireiro Sempre Bela tem uma tabela em que registra as quantidades de serviços executados nos "pes", nas "mãos" e o serviço de podologia das cinco manicures. Sabendo-se que cada uma ganha 50% do que faturou ao mês, criar um programa que possa calcular e imprimir quanto cada uma vai receber, uma vez que não têm carteiras assinadas, os valores, respectivamente, são: R\$10,00, R\$15,00 e R\$30,00.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int manicure=0, servico;
7     int servicos[5][3], i, j, soma;
8     //zerando a tabela de servicos
9     for (i=0; i<5; i++) {
10         for (j=0; j<3; j++)
11             servicos[i][j]=0;
12     }

```

```

13 //fazendo a leitura dos servicos
14 do {
15     printf("Informe a manicure <1-5>:\n");
16     scanf("%d", &manicure);
17     printf("Informe o servico <1-pe 2-mao 3-podologia>:\n");
18     scanf("%d", &servico);
19     if (manicure >= 1 && manicure <=5) {
20         if (servico >= 1 && servico <= 3)
21             servicos[manicure-1][servico-1]++;
22         else
23             printf("Servico invalido!\n");
24     }
25     else
26         printf("Manicure invalida!\n");
27 } while (manicure != 0);
28 //calculando e imprimindo o faturamento
29 printf("\nFaturamento das manicures:\n");
30 for (i=0; i<5; i++) {
31     soma = servicos[i][0] * 10;
32     soma += servicos[i][1] * 15;
33     soma += servicos[i][2] * 30;
34     printf("\nManicure %d: %d\n", i+1, soma);
35 }
36 }

```

14. Crie um programa que leia e armazene os elementos de uma matriz inteira com tamanho 5 X 5 e imprimi-la. Troque, a seguir:

- a segunda linha pela quinta;
- a terceira coluna pela quinta;
- a diagonal principal pela diagonal secundária.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int m[5][5];
7     int i, j, k, aux;
8
9     printf("\nDigite os 25 valores da matriz:\n");
10    for (i=0; i<5; i++)
11        for (j=0; j<5; j++)
12            scanf("%d", &m[i][j]);
13
14    //imprindo a matriz
15    for (i=0; i<5; i++) {
16        printf("\n");
17        for (j=0; j<5; j++)
18            printf("%d \t", m[i][j]);
19    }
20    printf("\n\n");
21

```

```

22 //trocando a segunda linha pela quinta
23 for (j=0; j<5; j++) {
24     aux = m[1][j];
25     m[1][j] = m[4][j];
26     m[4][j] = aux;
27 }
28 //imprimindo a matriz
29 for (i=0; i<5; i++) {
30     printf("\n");
31     for (j=0; j<5; j++)
32         printf("%d \t", m[i][j]);
33 }
34 printf("\n\n");
35
36 //trocando a terceira coluna pela quinta
37 for (i=0; i<5; i++) {
38     aux = m[i][2];
39     m[i][2] = m[i][4];
40     m[i][4] = aux;
41 }
42 //imprimindo a matriz
43 for (i=0; i<5; i++) {
44     printf("\n");
45     for (j=0; j<5; j++)
46         printf("%d \t", m[i][j]);
47 }
48 printf("\n\n");
49
50 //trocando a diagonal principal pela diagonal secundaria
51 i = 0;
52 k = 4;
53 for (j=0; j<5; j++) {
54     aux = m[i][j];
55     m[i][j] = m[k][j];
56     m[k][j] = aux;
57     k--;
58     i++;
59 }
60 //imprimindo a matriz
61 for (i=0; i<5; i++) {
62     printf("\n");
63     for (j=0; j<5; j++)
64         printf("%d \t", m[i][j]);
65 }
66 }

```

15. A matriz dados contém na 1ª coluna a matrícula do aluno; na 2ª, o sexo (0 para feminino e 1 para masculino); na 3ª, o código do curso, e na 4ª, o CR (coeficiente de rendimento).

Faça um programa que armazene esses dados sabendo-se que o código do curso é uma parte da matrícula: **aa** ano, **s** semestre, **ccc** código do curso e **nnn** matrícula no curso).

Além disso, um grupo empresarial resolveu premiar a aluna com CR mais alto de

um curso cujo código deverá ser digitado. Suponha 10 alunos e que o CR é um nº inteiro.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int dados[10][4], i, curso, crMaior=0, iMaior;
7     for (i=0; i<10; i++) {
8         printf("Informe os dados do aluno %d: \n", i+1);
9         printf("Informe a matricula <aasccnnn>:");
10        scanf("%d", &dados[i][0]);
11        printf("Informe o sexo <0-Feminino / 1-Masculino>:");
12        scanf("%d", &dados[i][1]);
13        printf("Informe o codigo do curso:");
14        scanf("%d", &dados[i][2]);
15        printf("Informe o CR:");
16        scanf("%d", &dados[i][3]);
17    }
18    //consultando a premiada
19    printf("Informe o curso para premiacao:\n");
20    scanf("%d", &curso);
21
22    for (i=0; i<10; i++) {
23        if (dados[i][2] == curso) {
24            if (dados[i][3] > crMaior && dados[i][1] == 0) {
25                crMaior = dados[i][3];
26                iMaior = i;
27            }
28        }
29    }
30    //imprimindo a aluna premiada
31    printf("Dados da aluna premiada\n");
32    printf("Matricula \t Sexo \t Curso \t CR \n");
33    printf("%d \t\t", dados[iMaior][0]);
34    printf("Fem. \t");
35    printf("%d \t", dados[iMaior][2]);
36    printf("%d \n", dados[iMaior][3]);
37 }

```

16. Faça um programa em C que possa armazenar as alturas de dez atletas de cinco delegações que participarão dos jogos de verão. Imprimir a maior altura de cada delegação.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int m[5][10], i, j, maior;
7
8     printf("\nDigite as 50 alturas dos atletas:\n");
9     for (i=0; i<5; i++) {
10        printf("Delegacao %d: \n", i+1);

```

```

11     for (j=0; j<10; j++)
12         scanf("%d", &m[i][j]);
13     }
14     printf("\nMaiores alturas entre as delegacoes:\n");
15     for (i=0; i<5; i++) {
16         maior = m[i][0];
17         for (j=1; j<10; j++) {
18             if (m[i][j] > maior)
19                 maior = m[i][j];
20         }
21         printf("\nMaior da delegacao %d: %d\n", i+1, maior);
22     }
23 }

```

17. A Viação José Maria Rodrigues tem na Rodoviária de Rio Novo uma tabela contendo os horários de partidas dos ônibus para Juiz de Fora nos sete dias da semana. Faça um programa que possa armazenar esses horários e os horários do dia quando forem solicitados pelo funcionário, sabendo-se que, no máximo, são dez horários. Ao final, o programa deve imprimir a lista de horários para todos os dias.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     float m[7][10], horario;
7     int dia, i, j, opcao;
8     int j1=0, j2=0, j3=0, j4=0, j5=0, j6=0, j7=0;
9     //zerando os horarios
10    for (i=0; i<7; i++) {
11        for (j=0; j<10; j++)
12            m[i][j]=0;
13    }
14    //cadastrando os horarios
15    do {
16        printf("Informe o dia <entre 1 e 7>:\n");
17        scanf("%d", &dia);
18        printf("Informe o horario de saida:\n");
19        scanf("%f", &horario);
20        switch (dia) {
21            case 1:
22                m[dia-1][j1] = horario;
23                j1++;
24                break;
25            case 2:
26                m[dia-1][j2] = horario;
27                j2++;
28                break;
29            case 3:
30                m[dia-1][j3] = horario;
31                j3++;
32                break;
33            case 4:

```

```

34         m[dia-1][j4] = horario;
35         j4++;
36         break;
37     case 5:
38         m[dia-1][j5] = horario;
39         j5++;
40         break;
41     case 6:
42         m[dia-1][j6] = horario;
43         j6++;
44         break;
45     case 7:
46         m[dia-1][j7] = horario;
47         j7++;
48         break;
49     default:
50         printf("Opcao invalida!\n");
51     }
52     if (j1 >= 10 & j2 >= 10 & j3 >= 10 & j4 >= 10 & j5 >= 10 & j6 >= 10
53         & j7 >= 10) {
54         printf("Horarios cheios!\n");
55         break;
56     }
57     printf("Continuar cadastrando <0-Parar / 1-Continuar?>\n");
58     scanf("%d", &opcao);
59 } while (opcao != 0);
60 //imprimindo os horarios
61 for (i=0; i<7; i++) {
62     printf("Dia %d \n", i+1);
63     for (j=0; j<10; j++)
64         printf("%f \t", m[i][j]);
65 }

```

18. Faça um programa que leia uma matriz 5 X 5 inteira e apresente uma determinada linha da matriz, solicitada via teclado.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int m[5][5], i, j, opcao;
7
8     printf("\nDigite os 25 valores da matriz:\n");
9     for (i=0; i<5; i++) {
10         for (j=0; j<5; j++)
11             scanf("%d", &m[i][j]);
12     }
13     printf("\nDigite a linha da matriz:\n");
14     scanf("%d", &opcao);
15     if (opcao > 0 && opcao < 6) {
16         opcao += -1;
17         printf("\nElementos da linha da matriz:\n");
18         for (i=0; i<5; i++)

```

```

19         printf("%d ", m[opcao][i]);
20     }
21     else
22         printf("\nA linha nao existe\n");
23 }

```

19. Construa um programa que carregue uma matriz 12 X 4 com os valores das vendas de uma loja, em cada linha represente um mês do ano, e cada coluna, uma semana do mês. Calcule e imprima:

- total vendido em cada mês do ano;
- total vendido em cada semana durante todo o ano;
- total vendido no ano.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     float m[12][4], total=0, mes=0;
7     int i, j, opcao;
8
9     for (i=0; i<12; i++) {
10        printf("\nInforme as vendas do mes %d:\n", i+1);
11        for (j=0; j<4; j++) {
12            printf("\nSemana %d:\n", j+1);
13            scanf("%f", &m[i][j]);
14        }
15    }
16    for (i=0; i<12; i++) {
17        mes = 0;
18        printf("Mes %d:\n", i+1);
19        for (j=0; j<4; j++) {
20            printf("Total vendido na semana %d: %f\n", j, m[i][j]);
21            mes += m[i][j];
22            total += m[i][j];
23        }
24        printf("\nTotal vendido no mes %d: %f\n", i+1, mes);
25    }
26    printf("\nTotal vendido no ano: %f\n", total);
27 }

```

20. Supondo que uma matriz apresente em cada linha o total de produtos vendidos ao mês por uma loja que trabalha com cinco tipos diferentes de produtos, construir um programa que leia esse total e, ao final, apresente o total de produtos vendidos em cada mês e o total de vendas por ano por produto.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {

```

```
6 float m[12][5], total=0, mes=0;
7 int i, j, opcao;
8
9 for (i=0; i<12; i++) {
10     printf("\nMes %d:\n", i+1);
11     for (j=0; j<5; j++) {
12         printf("Informe o total do produto %d\n", j+1);
13         scanf("%f", &m[i][j]);
14     }
15 }
16 for (i=0; i<12; i++) {
17     mes = 0;
18     for (j=0; j<5; j++) {
19         mes += m[i][j];
20         total += m[i][j];
21     }
22     printf("\nTotal vendido no mes %d: %f\n", i+1, mes);
23 }
24 printf("\nTotal vendido no ano: %f\n", total);
25 }
```

## Exercícios Resolvidos da Aula 6

1. Implemente um programa em C que leia o **nome**, a **idade** e o **endereço** de uma pessoa e armazene esses dados em uma estrutura. Em seguida, imprima na tela os dados da estrutura lida.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 struct pessoa {
5     char nome[30];
6     int idade;
7     char endereco[50];
8 };
9
10 void main()
11 {
12     struct pessoa p;
13     printf("Informe o nome: \n");
14     scanf("%s", &p.nome);
15     printf("Informe a idade: \n");
16     scanf("%d", &p.idade);
17     printf("Informe o endereço:\n");
18     scanf("%s", &p.endereco);
19
20     //imprimindo dados
21     printf("Nome: %s \n", p.nome);
22     printf("Idade: %d \n", p.idade);
23     printf("Endereço: %s \n", p.endereco);
24 }
```

2. Crie uma estrutura para representar as coordenadas de um **ponto** no plano (posições X e Y). Em seguida, declare e leia do teclado um ponto e exiba a distância dele até a origem das coordenadas, isto é, posição (0, 0). Para realizar o cálculo, utilize a fórmula a seguir:

$$d = \sqrt{(X_B - X_A)^2 + (Y_B - Y_A)^2} \quad (\text{E.1})$$

Em que:

- $d$  = distância entre os pontos A e B
- X = coordenada X em um ponto
- Y = coordenada Y em um ponto

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4
5 struct ponto {
6     int x;
7     int y;
8 };
9
10 void main()
11 {
12     struct ponto p;
13     float resultado;
14     printf("Informe o ponto x: \n");
15     scanf("%d", &p.x);
16     printf("Informe o ponto y: \n");
17     scanf("%d", &p.y);
18
19     resultado = sqrt(pow(p.x - 0, 2) + pow(p.y - 0, 2));
20
21     printf("Distancia do ponto de origem (0, 0): %f \n", resultado);
22 }

```

3. Crie uma estrutura para representar as coordenadas de um **ponto** no plano (posições X e Y). Em seguida, declare e leia do teclado dois pontos e exiba a distância entre eles, considere a mesma fórmula do exercício anterior.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4
5 struct ponto {
6     int x;
7     int y;
8 };
9
10 void main()
11 {
12     struct ponto pA, pB;
13     float resultado;
14     printf("Informe o ponto x da posicao A: \n");
15     scanf("%d", &pA.x);
16     printf("Informe o ponto y da posicao A: \n");
17     scanf("%d", &pA.y);
18
19     printf("Informe o ponto x da posicao B: \n");
20     scanf("%d", &pB.x);

```

```

21     printf("Informe o ponto y da posicao B: \n");
22     scanf("%d", &pB.y);
23
24     resultado = sqrt(pow(pB.x - pA.x, 2) + pow(pB.y - pA.y, 2));
25
26     printf("Distancia entre os pontos A e B: %f \n", resultado);
27 }

```

4. Cria uma estrutura chamada **retângulo**. Essa estrutura deverá conter o ponto superior esquerdo e o ponto inferior direito do retângulo. Cada ponto é definido por uma estrutura **Ponto**, a qual contém as posições X e Y. Faça um programa que declare e leia uma estrutura **retângulo** e exiba a área e o comprimento da diagonal e o perímetro desse retângulo.

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4
5  struct ponto {
6      int x;
7      int y;
8  };
9
10 void main()
11 {
12     struct ponto pA, pB;
13     float compDiagonal, altura, base, area, perimetro;
14     printf("Retangulo: X (ponto superior esquerdo) \n");
15     scanf("%d", &pA.x);
16     printf("Retangulo: Y (ponto superior esquerdo) \n");
17     scanf("%d", &pA.y);
18
19     printf("Retangulo: X (ponto inferior direito) \n");
20     scanf("%d", &pB.x);
21     printf("Retangulo: Y (ponto inferior direito) \n");
22     scanf("%d", &pB.y);
23
24     altura = sqrt(pow(pA.x - pA.x, 2) + pow(pA.y - pB.y, 2));
25     base = sqrt(pow(pB.x - pA.x, 2) + pow(pB.y - pB.y, 2));
26     area = altura * base;
27     compDiagonal = sqrt(pow(pB.x - pA.x, 2) + pow(pB.y - pA.y, 2));
28     perimetro = (altura + base) * 2;
29
30     printf("Comprimento da diagonal: %f \n", compDiagonal);
31     printf("Area: %f \n", area);
32     printf("Perimetro: %f \n", perimetro);
33 }

```

5. Usando a estrutura **retângulo** do exercício anterior, faça um programa que declare e leia uma estrutura **retângulo** e um **ponto**, e informe se esse ponto está ou não dentro do retângulo.

```

1  #include <stdio.h>
2  #include <stdlib.h>

```

```

3
4 struct ponto {
5     int x;
6     int y;
7 };
8
9 void main()
10 {
11     struct ponto pA, pB, p;
12     printf("Retangulo: X (ponto superior esquerdo) \n");
13     scanf("%d", &pA.x);
14     printf("Retangulo: Y (ponto superior esquerdo) \n");
15     scanf("%d", &pA.y);
16
17     printf("Retangulo: X (ponto inferior direito) \n");
18     scanf("%d", &pB.x);
19     printf("Retangulo: Y (ponto inferior direito) \n");
20     scanf("%d", &pB.y);
21
22     printf("Ponto: X \n");
23     scanf("%d", &p.x);
24     printf("Ponto: Y \n");
25     scanf("%d", &p.y);
26
27     if (p.x >= pA.x && p.x <= pB.x && p.y >= pB.y && p.y <= pA.y)
28         printf("O ponto esta dentro do retangulo\n");
29     else
30         printf("O ponto nao esta dentro do retangulo\n");
31 }

```

6. Crie uma estrutura representando um aluno de uma disciplina. Essa estrutura deve conter o número de matrícula do aluno, seu nome e as notas de três provas. Defina também um tipo para esta estrutura. Agora, escreva um programa que leia os dados de cinco alunos e os armazena nessa estrutura. Em seguida, exiba o nome e as notas do aluno que possui a maior média geral dentre os cinco.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 struct aluno {
5     int matricula;
6     char nome[30];
7     float p1, p2, p3;
8 };
9
10 typedef struct aluno Aluno;
11
12 void main()
13 {
14     Aluno alunos[5];
15     int i, iMaior=0;
16     float media, maiorMedia=0;
17
18     for (i=0; i<5; i++) {

```

```

19     printf("Informe a matricula do aluno: \n");
20     scanf("%d", &alunos[i].matricula);
21     printf("Informe o nome do aluno: \n");
22     scanf("%s", &alunos[i].nome);
23     printf("Informe a nota da P1: \n");
24     scanf("%f", &alunos[i].p1);
25     printf("Informe a nota da P2: \n");
26     scanf("%f", &alunos[i].p2);
27     printf("Informe a nota da P3: \n");
28     scanf("%f", &alunos[i].p3);
29     media = (alunos[i].p1 + alunos[i].p2 + alunos[i].p3) / 3;
30     if (media > maiorMedia) {
31         maiorMedia = media;
32         iMaior = i;
33     }
34 }
35
36 printf("-----Dados do aluno com maior media-----\n");
37 printf("Matricula: %d \n", alunos[iMaior].matricula);
38 printf("Nome.....: %s \n", alunos[iMaior].nome);
39 printf("Nota P1...: %f \n", alunos[iMaior].p1);
40 printf("Nota P2...: %f \n", alunos[iMaior].p2);
41 printf("Nota P3...: %f \n", alunos[iMaior].p3);
42 printf("Media.....: %f \n", maiorMedia);
43 }

```

7. Crie uma estrutura representando uma hora. Essa estrutura deve conter os campos hora, minuto e segundo. Agora, escreva um programa que leia um vetor de cinco posições dessa estrutura e imprima a maior hora.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 struct horario {
5     int hora;
6     int minutos;
7     int segundos;
8 };
9
10 typedef struct horario Hora;
11
12 void main()
13 {
14     Hora horarios[5], horaMaior;
15     int i;
16
17     horaMaior.hora = 0;
18     horaMaior.minutos = 0;
19     horaMaior.segundos = 0;
20
21     for (i=0; i<5; i++) {
22         printf("Informe a hora: \n");
23         scanf("%d", &horarios[i].hora);
24         printf("Informe os minutos: \n");

```

```

25     scanf("%d", &horarios[i].minutos);
26     printf("Informe os segundos: \n");
27     scanf("%d", &horarios[i].segundos);
28
29     if ((horarios[i].hora > horaMaior.hora) ||
30         (horarios[i].hora == horaMaior.hora && horarios[i].minutos >
31           horaMaior.minutos) ||
32         (horarios[i].hora == horaMaior.hora && horarios[i].minutos ==
33           horaMaior.minutos &&
34           horarios[i].segundos > horaMaior.segundos)) {
35         horaMaior.hora = horarios[i].hora;
36         horaMaior.minutos = horarios[i].minutos;
37         horaMaior.segundos = horarios[i].segundos;
38     }
39
40     printf("-----Maior hora lida-----\n");
41     printf("Hora.....: %d \n", horaMaior.hora);
42     printf("Minutos...: %d \n", horaMaior.minutos);
43     printf("Segundos.: %d \n", horaMaior.segundos);
44 }

```

8. Crie uma estrutura capaz de armazenar o nome e a data de nascimento de uma pessoa. Faça uso de estruturas aninhadas e definição de novo tipo de dado. Agora, escreva um programa que leia os dados de seis pessoas. Calcule e exiba os nomes da pessoa mais nova e da mais velha.

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  typedef struct dataNascimento {
5      int dia;
6      int mes;
7      int ano;
8  } DataNascimento;
9
10 typedef struct pessoa {
11     char nome[30];
12     DataNascimento nascimento;
13 } Pessoa;
14
15 void main()
16 {
17     Pessoa pessoas[6];
18     DataNascimento nascimentoMaior, nascimentoMenor;
19     nascimentoMaior.ano = 0;
20     nascimentoMenor.ano = 0;
21     int i, iMaior=0, iMenor=0;
22
23     for (i=0; i<6; i++) {
24         printf("Informe o nome: \n");
25         scanf("%s", &pessoas[i].nome);
26         printf("Informe o nascimento <Dia<enter>Mes<enter>Ano<enter>: \n");
27         scanf("%d", &pessoas[i].nascimento.dia);

```

```

28     scanf("%d", &personas[i].nascimento.mes);
29     scanf("%d", &personas[i].nascimento.ano);
30
31     //pessoa mais velha
32     if ((personas[i].nascimento.ano < nascimentoMenor.ano || i == 0) ||
33         (personas[i].nascimento.ano == nascimentoMenor.ano &&
34         pessoas[i].nascimento.mes < nascimentoMenor.mes) ||
35         (personas[i].nascimento.ano == nascimentoMenor.ano &&
36         pessoas[i].nascimento.mes == nascimentoMenor.mes &&
37         pessoas[i].nascimento.dia < nascimentoMenor.dia)) {
38         iMenor = i;
39         nascimentoMenor.dia = pessoas[i].nascimento.dia;
40         nascimentoMenor.mes = pessoas[i].nascimento.mes;
41         nascimentoMenor.ano = pessoas[i].nascimento.ano;
42     }
43
44     //pessoa mais jovem
45     if ((personas[i].nascimento.ano > nascimentoMaior.ano) ||
46         (personas[i].nascimento.ano == nascimentoMaior.ano &&
47         pessoas[i].nascimento.mes > nascimentoMaior.mes) ||
48         (personas[i].nascimento.ano == nascimentoMaior.ano &&
49         pessoas[i].nascimento.mes == nascimentoMaior.mes &&
50         pessoas[i].nascimento.dia > nascimentoMaior.dia)) {
51         iMaior = i;
52         nascimentoMaior.dia = pessoas[i].nascimento.dia;
53         nascimentoMaior.mes = pessoas[i].nascimento.mes;
54         nascimentoMaior.ano = pessoas[i].nascimento.ano;
55     }
56 }
57
58 printf("-----Pessoa mais velha-----\n");
59 printf("Nome.....: %s \n", pessoas[iMenor].nome);
60 printf("Nascimento..: %d/%d/%d \n", pessoas[iMenor].nascimento.dia,
61         pessoas[iMenor].nascimento.mes,
62         pessoas[iMenor].nascimento.ano);
63
64 printf("-----Pessoa mais jovem-----\n");
65 printf("Nome.....: %s \n", pessoas[iMaior].nome);
66 printf("Nascimento..: %d/%d/%d \n", pessoas[iMaior].nascimento.dia,
67         pessoas[iMaior].nascimento.mes,
68         pessoas[iMaior].nascimento.ano);
69 }

```

9. Crie uma estrutura representando um atleta. Essa estrutura deve conter o nome do atleta, seu esporte, idade e altura. Agora, escreva um programa que leia os dados de cinco atletas. Calcule e exiba os nomes do atleta mais alto e do mais velho.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 struct atleta {
5     char nome[30];
6     char esporte[30];

```

```

7     int idade;
8     float altura;
9 };
10
11 typedef struct atleta Atleta;
12
13 void main()
14 {
15     Atleta atletas[5];
16     int atletaVelho=0, i, iVelho, iAlto;
17     float atletaAlto=0;
18
19     for (i=0; i<5; i++) {
20         printf("Informe o nome: \n");
21         scanf("%s", &atletas[i].nome);
22         printf("Informe o esporte: \n");
23         scanf("%s", &atletas[i].esporte);
24         printf("Informe a idade: \n");
25         scanf("%d", &atletas[i].idade);
26         printf("Informe a altura: \n");
27         scanf("%f", &atletas[i].altura);
28
29         //atleta mais velho
30         if (atletas[i].idade > atletaVelho) {
31             iVelho = i;
32             atletaVelho = atletas[i].idade;
33         }
34
35         //atleta mais alto
36         if (atletas[i].altura > atletaAlto) {
37             iAlto = i;
38             atletaAlto = atletas[i].altura;
39         }
40     }
41
42     printf("-----Atleta mais velho-----\n");
43     printf("Nome...: %s \n", atletas[iVelho].nome);
44     printf("Idade...: %d \n", atletas[iVelho].idade);
45
46     printf("-----Atleta mais alto-----\n");
47     printf("Nome...: %s \n", atletas[iAlto].nome);
48     printf("Altura...: %f \n", atletas[iAlto].altura);
49 }

```

10. Usando a estrutura "atleta" do exercício anterior, escreva um programa que leia os dados de cinco atletas e os exiba por ordem de idade, do mais velho para o mais novo.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 struct atleta {
6     char nome[30];

```

```
7     char esporte[30];
8     int idade;
9     float altura;
10 };
11
12 typedef struct atleta Atleta;
13
14 void main()
15 {
16     Atleta atletas[5], aux;
17     int i, j;
18
19     for (i=0; i<5; i++) {
20         printf("Informe o nome: \n");
21         scanf("%s", &atletas[i].nome);
22         printf("Informe o esporte: \n");
23         scanf("%s", &atletas[i].esporte);
24         printf("Informe a idade: \n");
25         scanf("%d", &atletas[i].idade);
26         printf("Informe a altura: \n");
27         scanf("%f", &atletas[i].altura);
28     }
29
30     for (i=0; i<5; i++) {
31         for (j=i+1; j<5; j++) {
32             if (atletas[i].idade < atletas[j].idade) {
33                 //salva na variavel auxiliar
34                 strcpy(aux.nome, atletas[i].nome);
35                 strcpy(aux.esporte, atletas[i].esporte);
36                 aux.idade = atletas[i].idade;
37                 aux.altura = atletas[i].altura;
38
39                 //trocando as posicoes
40                 strcpy(atletas[i].nome, atletas[j].nome);
41                 strcpy(atletas[i].esporte, atletas[j].esporte);
42                 atletas[i].idade = atletas[j].idade;
43                 atletas[i].altura = atletas[j].altura;
44
45                 strcpy(atletas[j].nome, aux.nome);
46                 strcpy(atletas[j].esporte, aux.esporte);
47                 atletas[j].idade = aux.idade;
48                 atletas[j].altura = aux.altura;
49             }
50         }
51     }
52
53     for (i=0; i<5; i++) {
54         printf("-----Atletas-----\n");
55         printf("Nome.....: %s \n", atletas[i].nome);
56         printf("Esporte..: %s \n", atletas[i].esporte);
57         printf("Idade....: %d \n", atletas[i].idade);
58         printf("Altura...: %f \n", atletas[i].altura);
59     }
60 }
```

11. Escreva um programa que contenha uma estrutura representando uma data válida. Essa estrutura deve conter os campos dia, mês e ano. Em seguida, leia duas datas e armazene nessa estrutura. Calcule e exiba o número de dias que decorrem entre as duas datas.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 typedef struct data {
5     int dia;
6     int mes;
7     int ano;
8 } Data;
9
10 void main()
11 {
12     Data data1, data2;
13     int dias;
14
15     printf("Informe a primeira data <Dia<enter>Mes<enter>Ano<enter>: \n");
16     scanf("%d", &data1.dia);
17     scanf("%d", &data1.mes);
18     scanf("%d", &data1.ano);
19
20     printf("Informe a segunda data <Dia<enter>Mes<enter>Ano<enter>: \n");
21     scanf("%d", &data2.dia);
22     scanf("%d", &data2.mes);
23     scanf("%d", &data2.ano);
24
25     dias = (data2.ano - data1.ano - 1) * 365;
26     dias += ((12 - data1.mes) * 30) + (data2.mes * 30);
27     dias += data1.dia + data2.dia;
28
29     printf("Total de dias: %d \n", dias);
30 }
```

12. Astolfov Oliveirescu é técnico de um time da série C do poderoso campeonato de futebol profissional da Albânia. Ele deseja manter os dados dos seus jogadores guardados de forma minuciosa. Ajude-o fazendo um programa para armazenar os seguintes dados de cada jogador: nº da camisa, peso (kg), altura (m) e a posição em que joga (atacante, defensor ou meio campista). Lembre-se que o time tem 22 jogadores, entre reservas e titulares. Leia os dados e depois gere um relatório no vídeo, devidamente tabulado/formatado.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 struct tipo_dados_time{
5     int num_camisa;
6     float altura;
7     float peso;
8     char posicao[20];
9 }time[22];
10
```

```

11 void main()
12 {
13     int i;
14
15     //coletando dados
16     printf("-----Digite os dados abaixo-----\n\n");
17     for(i=0; i<22; i++){
18         printf("Digite o numero da camisa: ");
19         scanf("%d", &time[i].num_camisa);
20         printf("Digite a altura: ");
21         scanf("%f", &time[i].altura);
22         printf("Digite o peso: ");
23         scanf("%f%c", &time[i].peso);
24         printf("Digite a posicao: ");
25         fgets(time[i].posicao, 20, stdin);
26     }
27
28     //imprimindo dados coletados tabulados
29     printf("Relatorio:\n");
30     for(i=0; i<22; i++)
31         printf("Camisa n: %d | Peso: %2.f | Altura: %2.f | Posicao: %s \n",
32                time[i].num_camisa, time[i].peso,
33                stime[i].altura, time[i].posicao);
34 }

```

13. Um clube social com 37 associados deseja que você faça um programa para armazenar os dados cadastrais desses associados. Os dados são: nome, dia, mês e ano de nascimento, valor da mensalidade e quantidade de dependentes. O programa deverá ler os dados e imprimir depois na tela. Deverá também informar o associado (ou os associados) com o maior número de dependentes.

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  struct tipo_data{
5      int dia;
6      int mes;
7      int ano;
8  };
9
10 struct tipo_dados_climat{
11     char nome[30];
12     struct tipo_data nascimento;
13     float mensalidade;
14     int dependentes;
15 }clube[37];
16
17 void main()
18 {
19     int i, assoc, aux;
20
21     for(i=0; i<37; i++){
22         printf("Informe o nome: ");
23         gets(clube[i].nome);

```

```

24     printf("Informe o dia do seu nascimento: ");
25     scanf("%d", &clube[i].nascimento.dia);
26     printf("Informe o mes do seu nascimento: ");
27     scanf("%d", &clube[i].nascimento.mes);
28     printf("Informe o ano do seu nascimento: ");
29     scanf("%d", &clube[i].nascimento.ano);
30     printf("Informe o valor da mensalidade: ");
31     scanf("%f", &clube[i].mensalidade);
32     printf("Informe o numero de dependentes: ");
33     scanf("%d%c", &clube[i].dependentes);
34 }
35
36 //comparando valores do vetor para verificar o maior
37 aux = clube[0].dependentes;
38 assoc = 0;
39
40 for(i=1; i<37; i++){
41     if (aux <= clube[i].dependentes){
42         aux = clube[i].dependentes;
43         assoc = i;
44     }
45     if (aux >= clube[i].dependentes){
46         aux = aux;
47         assoc = assoc;
48     }
49 }
50
51 //imprimindo os resultados
52 printf("Associado\tNascimento\tMensalidade (R$)\tDep\n");
53 for(i=0; i<37; i++) {
54     printf("%s\t", clube[i].nome);
55     printf("%d/%d/%d\t", clube[i].nascimento.dia,
56         clube[i].nascimento.mes, clube[i].nascimento.ano);
57     printf("%.2f\t", clube[i].mensalidade);
58     printf("%d\n", clube[i].dependentes);
59 }
60 printf("O associado com maior numero de dependentes e %s, com %d
61     clube[assoc].nome,
62     clube[assoc].dependentes);
63 }

```

14. Crie um programa que tenha uma estrutura para armazenar o nome, a idade e número da carteira de sócio de 50 associados de um clube. Crie também uma estrutura, dentro desta anterior, chamada **dados** que contenha o endereço, telefone e data de nascimento.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 struct dados{
5     char endereco[50];
6     int telefone;
7     char nascimento[12];

```

```

8  };
9
10 struct clube{
11     char nome[30];
12     int idade;
13     int carteira;
14     struct dados data;
15
16 }clube[50];
17
18 void main()
19 {
20     int i;
21
22     //le as informacoes
23     for(i=0; i<50; i++){
24         printf("Digite o nome: ");
25         gets(clube[i].nome);
26         printf("Digite a idade: ");
27         scanf("%d", &clube[i].idade);
28         printf("Digite o numero da carteira de socio: ");
29         scanf("%d%c", &clube[i].carteira);
30         printf("Digite o endereco: ");
31         gets(clube[i].data.endereco);
32         printf("Digite o telefone: ");
33         scanf("%d%c", &clube[i].data.telefone);
34         printf("Digite sua data de nascimento: ");
35         gets(clube[i].data.nascimento);
36     }
37     printf("Nome\tIdade\tCarteira\tEnd\tTel\tNasc\n");
38     //escreve na saida padrao
39     for(i=0; i<50; i++){
40         printf("%s\t", clube[i].nome);
41         printf("%d\t", clube[i].idade);
42         printf("%d\t", clube[i].carteira);
43         printf("%s\t", clube[i].data.endereco);
44         printf("%d\t", clube[i].data.telefone);
45         printf("%s\n", clube[i].data.nascimento);
46     }
47 }

```

15. Crie um programa com uma estrutura para simular uma agenda de telefone celular, com até 100 registros. Nessa agenda deve constar o nome, sobrenome, número de telefone móvel, número de telefone fixo e e-mail. O programa deverá fazer a leitura e, após isso, mostrar os dados na tela.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 struct telefone{
5     char nome[30];
6     char sobrenome[30];
7     char movel[30];
8     char fixo[30];

```

```
9     char email[30];
10 };
11
12 void main()
13 {
14     struct telefone p[100];
15     int i;
16
17     for(i=0; i<3; i++){
18         printf("Digite o nome: ");
19         gets(p[i].nome);
20         printf("Digite o sobrenome: ");
21         gets(p[i].sobrenome);
22         printf("Digite o numero celular: ");
23         gets(p[i].movel);
24         printf("Digite o numero do tel fixo: ");
25         gets(p[i].fixo);
26         printf("Digite o e-mail: ");
27         gets(p[i].email);
28     }
29
30     for(i=0; i<3; i++){
31         printf("Nome: %s\n", p[i].nome);
32         printf("Sobrenome: %s\n", p[i].sobrenome);
33         printf("Celular: %s\n", p[i].movel);
34         printf("Fixo: %s\n", p[i].fixo);
35         printf("E-mail: %s\n", p[i].email);
36         printf("\n");
37     }
38 }
```

## Exercícios Resolvidos da Aula 7

1. Faça um programa em C que leia três números e, para cada um, imprimir o dobro. O cálculo deverá ser realizado por uma função e o resultado impresso ao final do programa.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void dobraNumero(int n1, int n2, int n3);
5
6 void main()
7 {
8     int n1, n2, n3;
9
10    printf("\nDigite os 3 numeros:");
11    scanf("%d %d %d", &n1, &n2, &n3);
12
13    dobraNumero(n1, n2, n3);
14 }
15
16 void dobraNumero(int n1, int n2, int n3) {
17     printf("\nDobro do numero 1: %d", n1 * 2);
18     printf("\nDobro do numero 2: %d", n2 * 2);
19     printf("\nDobro do numero 3: %d", n3 * 2);
20 }
```

2. Faça um programa que receba as notas de três provas e calcule a média. Para o cálculo, escreva uma função. O programa deve imprimir a média ao final.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 float calculaMedia(float n1, float n2, float n3) {
5     return ((n1 + n2 + n3) / 3);
6 }
7
8 void main()
9 {
10    float n1, n2, n3, media = 0;
```

```

11
12     printf("\nDigite as 3 notas:");
13     scanf("%f %f %f", &n1, &n2, &n3);
14
15     media = calculaMedia(n1, n2, n3);
16     printf("\nMedia final: %f", media);
17 }

```

3. Faça um programa em C que leia o valor de um ângulo em graus e o converta, utilizando uma função, para radianos e ao final imprima o resultado. Veja a fórmula de cálculo a seguir.

$$rad = \frac{ang \times pi}{180} \quad (G.1)$$

Em que:

- rad = ângulo em radianos
- ang = ângulo em graus
- pi = número do pi

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 float calculaRadiano(int angGrau) {
5     return (angGrau * 3.14) / 180;
6 }
7
8 void main()
9 {
10     int anguloGrau;
11     float radianos = 0;
12     printf("\nDigite o valor do angulo em graus:");
13     scanf("%d", &anguloGrau);
14
15     radianos = calculaRadiano(anguloGrau);
16     printf("\nAngulo em radianos: %.2f", radianos);
17 }

```

4. Faça um programa que calcule e imprima o fatorial de um número, usando uma função que receba um valor e retorne o fatorial desse valor.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 float fatorial(int n) {
5     int fat=1, i;
6
7     for (i=2; i <= n; i++)
8         fat *= i;
9
10    return fat;
11 }

```

```

12
13 void main()
14 {
15     int n, fat=0;
16
17     printf("\nDigite o numero para calcular o fatorial:");
18     scanf("%d", &n);
19
20     fat = fatorial(n);
21     printf("\nfatorial do numero %d: %d", n, fat);
22 }

```

5. Faça um programa que verifique se um número é primo por meio de um função. Ao final imprima o resultado.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int numeroPrimo(int n) {
5     int i, aux=0;
6     for(i=1; i <= n; i++)
7         if(n%i == 0)
8             aux++;
9     return aux;
10 }
11
12 void main()
13 {
14     int n, primo;
15     printf("\nDigite o numero: ");
16     scanf("%d", &n);
17
18     primo = numeroPrimo(n);
19     if (primo == 2)
20         printf("\nNumero primo");
21     else
22         printf("\nNumero nao primo");
23 }

```

6. Faça um programa que leia o saldo e o % de reajuste de uma aplicação financeira e imprimir o novo saldo após o reajuste. O cálculo deve ser feito por uma função.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int reajusteSaldo(float saldo, float reajuste) {
5     return saldo + (saldo * (reajuste / 100));
6 }
7
8 void main()
9 {
10     float saldo, reajuste, novoSaldo=0;
11
12     printf("\nDigite o saldo:");

```

```

13  scanf("%f", &saldo);
14  printf("\nDigite o percentual de reajuste:");
15  scanf("%f", &reajuste);
16
17  novoSaldo = reajusteSaldo(saldo, reajuste);
18  printf("\nNovo saldo com reajuste: %f", novoSaldo);
19  }

```

7. Faça um programa que leia a base e a altura de um retângulo e imprima o perímetro, a área e a diagonal. Para fazer os cálculos, implemente três funções, cada uma deve realizar um cálculo específico conforme solicitado. Utilize as fórmulas a seguir.

$$\textit{perimetro} = 2 \times (\textit{base} + \textit{altura}) \quad (\text{G.2})$$

$$\textit{area} = \textit{base} \times \textit{altura} \quad (\text{G.3})$$

$$\textit{diagonal} = \sqrt{\textit{base}^2 + \textit{altura}^2} \quad (\text{G.4})$$

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4
5  float perimetro(int base, int altura) {
6      return 2 * (base + altura);
7  }
8
9  float area(int base, int altura) {
10     return base * altura;
11 }
12
13 float diametro(int base, int altura) {
14     float aux = base * base + altura * altura;
15     aux = sqrt(aux);
16     return aux;
17 }
18
19 void main()
20 {
21     int base, altura;
22     float per, are, dia;
23
24     printf("\nDigite a base do retangulo: ");
25     scanf("%d", &base);
26     printf("\nDigite a altura do retangulo: ");
27     scanf("%d", &altura);
28
29     per = perimetro(base, altura);
30     are = area(base, altura);
31     dia = diametro(base, altura);
32

```

```

33     printf("\nPerimetro: %f", per);
34     printf("\nBase: %f", are);
35     printf("\nDiametro: %f", dia);
36 }

```

8. Faça um programa que leia o raio de um círculo e imprima o perímetro e a área. Para fazer os cálculos, implemente duas funções, cada uma deve realizar um cálculo específico conforme solicitado. Utilize as fórmulas a seguir.

$$perimetro = 2 \times pi \times raio \quad (G.5)$$

$$area = pi \times raio^2 \quad (G.6)$$

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 float perimetro(int raio) {
5     return 2 * 3.14 * raio;
6 }
7
8 float area(int raio) {
9     return 3.14 * raio * raio;
10 }
11
12 void main()
13 {
14     float raio, per, are;
15
16     printf("\nDigite o raio do círculo: ");
17     scanf("%f", &raio);
18
19     per = perimetro(raio);
20     are = area(raio);
21
22     printf("\nPerimetro: %f", per);
23     printf("\nBase: %f", are);
24 }

```

9. Faça um programa que leia o lado de um quadrado e imprima o perímetro, a área e a diagonal. Para fazer o cálculo, implemente três funções, cada uma deve realizar um cálculo específico conforme solicitado. Utilize as fórmulas a seguir.

$$perimetro = 4 \times lado \quad (G.7)$$

$$area = lado^2 \quad (G.8)$$

$$diagonal = lado \times \sqrt{2} \quad (G.9)$$

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4
5  float perimetro(int lado) {
6      return 4 * lado;
7  }
8
9  float area(int lado) {
10     return lado * lado;
11 }
12
13 float diagonal(int lado) {
14     return lado * sqrt(2);
15 }
16
17 void main()
18 {
19     float lado, per, are, dia;
20
21     printf("\nDigite o lado do quadrado: ");
22     scanf("%f", &lado);
23
24     per = perimetro(lado);
25     are = area(lado);
26     dia = diagonal(lado);
27
28     printf("\nPerimetro: %f", per);
29     printf("\nBase: %f", are);
30     printf("\nDiagonal: %f", dia);
31 }

```

10. Faça um programa que leia os lados **a**, **b** e **c** de um paralelepípedo e imprima a diagonal. Para fazer o cálculo, implemente uma função. Utilize a fórmula a seguir.

$$diagonal = \sqrt{a^2 + b^2 + c^2} \quad (G.10)$$

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4
5  float diagonal(float a, float b, float c) {
6      return sqrt(a * a + b * b + c * c);
7  }
8
9  void main()
10 {
11     float a, b, c, dia;
12
13     printf("\nDigite os 3 lados do paralelepipedo: ");
14     scanf("%f %f %f", &a, &b, &c);
15
16     dia = diagonal(a, b, c);

```

```

17
18     printf("\nDiagonal: %f", dia);
19 }

```

11. Faça um programa que leia a diagonal maior e a diagonal menor de um losango e imprima a área. Para fazer o cálculo, implemente uma função. Utilize a fórmula a seguir.

$$area = \frac{(diagonalMaior \times diagonalMenor)}{2} \quad (G.11)$$

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 float area(float diagonalMaior, float diagonalMenor) {
5     return (diagonalMaior * diagonalMenor) / 2;
6 }
7
8 void main()
9 {
10     float diagonalMaior, diagonalMenor, are;
11
12     printf("\nDigite as diagonais maior e menor do losango: ");
13     scanf("%f %f", &diagonalMaior, &diagonalMenor);
14
15     are = area(diagonalMaior, diagonalMenor);
16
17     printf("\nArea: %f", are);
18 }

```

12. Faça um programa que leia os catetos (dois catetos) de um triângulo retângulo e imprima a hipotenusa. Para fazer o cálculo, implemente uma função. Utilize a fórmula a seguir.

$$hipotenusa = \sqrt{cateto1^2 + cateto2^2} \quad (G.12)$$

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4
5 float hipotenusa(float diagonalMaior, float diagonalMenor) {
6     return sqrt(diagonalMaior * diagonalMaior + diagonalMenor *
7         diagonalMenor);
8 }
9
10 void main()
11 {
12     float catetoOposto, catetoAdjacente, hipo;
13
14     printf("\nDigite os catetos: ");
15     scanf("%f %f", &catetoOposto, &catetoAdjacente);

```

```
16     hipo = hipotenusa(catetoOposto, catetoAdjacente);
17
18     printf("\nHipotenusa: %f", hipo);
19 }
```

13. Em épocas de pouco dinheiro, os comerciantes estão procurando aumentar suas vendas oferecendo desconto. Faça um programa que permita entrar com o valor de um produto e o percentual de desconto e imprimir o novo valor com base no percentual informado. Para fazer o cálculo, implemente uma função.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int reajusteSaldo(float valor, float reajuste) {
5     return valor + (valor * (reajuste / 100));
6 }
7
8 void main()
9 {
10     float valor, reajuste, novoSaldo=0;
11
12     printf("\nDigite o valor:");
13     scanf("%f", &valor);
14     printf("\nDigite o percentual de reajuste:");
15     scanf("%f", &reajuste);
16
17     novoSaldo = reajusteSaldo(valor, reajuste);
18     printf("\nNovo saldo com reajuste: %f", novoSaldo);
19 }
```

14. Faça um programa que verifique quantas vezes um número é divisível por outro. A função deve receber dois parâmetros, o dividendo e o divisor. Ao ler o divisor, é importante verificar se ele é menor que o dividendo. Ao final imprima o resultado.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int divisao(int dividendo, int divisor) {
5     int cont = 0;
6     while (dividendo >= divisor) {
7         dividendo = dividendo / divisor;
8         cont++;
9     }
10    return cont;
11 }
12
13 void main()
14 {
15     int dividendo, divisor, div;
16
17     printf("\nDigite o dividendo:");
18     scanf("%d", &dividendo);
19     printf("\nDigite o divisor:");
20     scanf("%d", &divisor);
```

```
21
22     if (divisor > dividendo)
23         printf("\nDivisor maior que dividendo");
24     else {
25         div = divisao(dividendo, divisor);
26         printf("\nNumero de divisoes: %d", div);
27     }
28 }
```

15. Construa um programa em C que leia um caractere (letra) e, por meio de uma função, retorne se este caractere é uma consoante ou uma vogal. Ao final imprima o resultado.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int letra(char c)
5 {
6     switch (c) {
7     case 'a':
8         return 1;
9         break;
10    case 'e':
11        return 1;
12        break;
13    case 'i':
14        return 1;
15        break;
16    case 'o':
17        return 1;
18        break;
19    case 'u':
20        return 1;
21        break;
22    default:
23        return 0;
24    }
25 }
26
27 void main()
28 {
29     char c;
30     int i;
31
32     printf("\nDigite o caracter para verificar:");
33     scanf("%c", &c);
34
35     i = letra(c);
36     if (i == 1)
37         printf("\nVogal");
38     else
39         printf("\nConsoante");
40 }
```

16. Construa um programa que leia um valor inteiro e retorne se a raiz desse número é exata ou não. Escreva uma função para fazer a validação. Ao final imprima o resultado.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4
5 int raiz(int n) {
6     float aux = sqrt(n);
7     int a = aux;
8     aux = aux - a;
9     if (aux > 0)
10        return 0;
11    else
12        return 1;
13 }
14
15 void main()
16 {
17     int n, i;
18
19     printf("\nDigite o numero para verificar a raiz:");
20     scanf("%d", &n);
21
22     i = raiz(n);
23
24     if (i == 1)
25         printf("\nRaiz inteira");
26     else
27         printf("\nRaiz nao inteira");
28 }
```

17. Implemente um programa que leia uma mensagem e um caractere. Após a leitura, o programa deve, por meio de função, retirar todas as ocorrências do caractere informado na mensagem colocando \* em seu lugar. A função deve também retornar o total de caracteres retirados. Ao final, o programa deve imprimir a frase ajustada e a quantidade de caracteres substituídos.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 int contaLetra(char c[100], char caracter) {
6     int i, cont=0;
7     for (i=0; i < strlen(c); i++) {
8         if (c[i] == caracter) {
9             c[i] = '*';
10            cont++;
11        }
12    }
13
14    for (i=0; i < strlen(c); i++)
15        printf("%c", c[i]);
16 }
```

```
17     return cont;
18 }
19
20 void main()
21 {
22     char c[100], caracter;
23
24     printf("\nDigite a frase:");
25     gets(c);
26     printf("\nDigite o caracter:");
27     scanf("%c", &caracter);
28
29     int cont = contaLetra(c, caracter);
30
31     printf("\nQuantidade de caracteres encontrados: %d", cont);
32 }
```

18. Faça um programa que leia um vetor com tamanho 10 de números inteiros. Após ler, uma função deve verificar se o vetor está ordenado, de forma crescente ou decrescente, ou se não está ordenado. Imprimir essa resposta no final do programa.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int vetor(int *v) {
5     int cres=1, decres=1, i, j;
6
7     for (i=0; i < 9; i++) {
8         if (v[i] > v[i+1])
9             cres = 0;
10        else if (v[i] < v[i+1])
11            decres = 0;
12    }
13
14    if (cres == 1)
15        return 1;
16    else if (decres == 1)
17        return 2;
18    else
19        return 0;
20 }
21
22 void main()
23 {
24     int v[10], i, resultado;
25
26     printf("\nDigite os 10 elementos do vetor:");
27     for (i=0; i<10; i++)
28         scanf("%d", &v[i]);
29
30     resultado = vetor(v);
31
32     if (resultado == 1)
```

```

33     printf("\nVetor ordenado de forma crescente");
34     else if (resultado == 2)
35         printf("\nVetor ordenado de forma decrescente");
36     else
37         printf("\nVetor nao ordenado");
38 }

```

19. Faça um programa que leia um vetor com tamanho 10 de números inteiros. Após ler, uma função deve criar um novo vetor com base no primeiro, mas, o novo vetor deve ser ordenado de forma crescente. O programa deve imprimir este novo vetor após a ordenação.

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void vetor(int *v) {
5      int cres[10], i, j, aux;
6
7      for (i=0; i < 10; i++)
8          cres[i] = v[i];
9
10     for (i=0; i < 9; i++) {
11         for (j=i+1; j < 10; j++) {
12             if (cres[i] > cres[j]) {
13                 aux = cres[i];
14                 cres[i] = cres[j];
15                 cres[j] = aux;
16             }
17         }
18     }
19
20     printf("\nElementos do vetor ordenado");
21     for (i=0; i<10; i++)
22         printf("%d", cres[i]);
23 }
24
25 void main()
26 {
27     int v[10], i;
28
29     printf("\nDigite os 10 elementos do vetor:");
30     for (i=0; i<10; i++)
31         scanf("%d", &v[i]);
32
33     vetor(v);
34 }

```

20. Faça um programa que leia 20 de números inteiros e armazene em um vetor. Após essa leitura, o programa deve ler um novo número inteiro para ser buscado no vetor. Uma função deve verificar se o número lido por último está no vetor e retornar a posição do número no vetor, caso esteja, ou -1, caso não esteja.

```

1  #include <stdio.h>
2  #include <stdlib.h>

```

```
3
4 int vetor(int *v, int num) {
5     int i;
6
7     for (i=0; i<20; i++) {
8         if (v[i] == num)
9             return i;
10    }
11    return -1;
12 }
13
14 void main()
15 {
16     int v[20], i, num;
17
18     printf("\nDigite os 20 elementos do vetor:");
19     for (i=0; i<20; i++)
20         scanf("%d", &v[i]);
21     printf("\nDigite o elemento para buscar no vetor:");
22     scanf("%d", &num);
23
24     i = vetor(v, num);
25     if (i != -1)
26         printf("\nPosicao do elemento no vetor: %d", i);
27     else
28         printf("\nElemento nao consta no vetor");
29 }
```



## Exercícios Resolvidos da Aula 8

1. Faça um programa em C que calcule, por meio de uma função recursiva,  $a \times b$  usando a adição, em que **a** e **b** são inteiros não-negativos.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 float calculaMult(int a, int b) {
5     if (b > 0)
6         return a + calculaMult(a, b-1);
7     else
8         return 0;
9 }
10
11 void main()
12 {
13     int a, b;
14     float resultado;
15     printf("Informe o valor de A:\n");
16     scanf("%d", &a);
17     printf("Informe o valor de B:\n");
18     scanf("%d", &b);
19
20     resultado = calculaMult(a, b);
21
22     printf("Resultado da multiplicacao por soma: %f \n", resultado);
23 }
```

2. Crie uma função recursiva que receba um número inteiro positivo **N** e calcule o somatório dos números de 1 a **N**.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int soma(int k) {
5     if (k > 0)
6         return k + soma(k-1);
7     else
8         return k;
```

```
9 }
10
11 void main()
12 {
13     int numero, resultado;
14     printf("Digite um numero inteiro positivo:");
15     scanf("%d", &numero);
16
17     resultado = soma(numero);
18
19     printf("Soma de 1 a %d: %d", numero, resultado);
20 }
```

3. Considere um vetor **vet** de tamanho 20. Construa um programa com algoritmos recursivos para calcular:

- o elemento máximo do vetor;
- o elemento mínimo do vetor;
- a soma dos elementos do vetor;
- o produto dos elementos do vetor;
- a média dos elementos do vetor.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 //retorna o maior elemento do vetor
5 int maxVet(int *vet, int tam) {
6     if (tam == 1)
7         return vet[0];
8     else {
9         int aux;
10        aux = maxVet(vet, tam-1);
11        if (aux > vet[tam-1])
12            return aux;
13        else
14            return (vet[tam-1]);
15    }
16 }
17
18 //retorna o menor elemento do vetor
19 int minVet(int *vet, int tam) {
20     if (tam == 1)
21         return vet[0];
22     else {
23         int aux;
24         aux = minVet(vet, tam-1);
25         if (aux < vet[tam-1])
26             return aux;
27         else
28             return (vet[tam-1]);
29     }
30 }
```

```
31
32 //retorna a soma dos elementos do vetor
33 int somaVet(int *vet, int ultPos) {
34     if (ultPos > 0)
35         return vet[ultPos] + somaVet(vet, ultPos-1);
36     else
37         return vet[ultPos];
38 }
39
40 //retorna o produto dos elementos do vetor
41 int multVet(int *vet, int ultPos) {
42     if (ultPos > 0)
43         return vet[ultPos] * multVet(vet, ultPos-1);
44     else
45         return vet[ultPos];
46 }
47
48 //retorna a media dos elementos do vetor
49 float mediaVet(int *vet, int ultPos, int tam) {
50     if (ultPos == tam-1)
51         return (vet[ultPos] + mediaVet(vet, ultPos-1, tam)) / tam;
52     if (ultPos > 0 && ultPos != tam-1)
53         return vet[ultPos] + mediaVet(vet, ultPos-1, tam);
54     else
55         return vet[ultPos];
56 }
57
58 void main()
59 {
60     int vet[20], i, max, min;
61     int soma, prod;
62     float media;
63
64     for (i=0; i<20; i++) {
65         printf("Informe um valor inteiro %d: \n", i+1);
66         scanf("%d", &vet[i]);
67     }
68
69     max = maxVet(vet, 20);
70     min = minVet(vet, 20);
71     soma = somaVet(vet, 19);
72     prod = multVet(vet, 19);
73     media = mediaVet(vet, 19, 20);
74     printf("Maior: %d \n", max);
75     printf("Menor: %d \n", min);
76     printf("Soma: %d \n", soma);
77     printf("Produto: %d \n", prod);
78     printf("Media: %f \n", media);
79 }
```

4. A sequência de Fibonacci é a sequência de inteiros: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, .... Implemente uma função recursiva que calcule e imprima todos os elementos da série Fibonacci de 0 até **n**. Em que, **n** deve ser informado pelo usuário do programa.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int fibonacci(int num) {
5     if(num==1 || num==2)
6         return 1;
7     else
8         return fibonacci(num-1) + fibonacci(num-2);
9 }
10
11 void main()
12 {
13     int n,i;
14     printf("Digite a quantidade de termos da sequencia de Fibonacci: ");
15     scanf("%d", &n);
16     printf("\nA sequencia de Fibonacci e: 0 ");
17     for(i=0; i<n; i++)
18         printf("%d ", fibonacci(i+1));
19 }

```

5. Escreva uma função recursiva em C para calcular o máximo divisor comum de dois números,  $\text{mdc}(x, y)$ .

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int mdc(int m, int n) {
5     if (n==0)
6         return m;
7     return mdc(n, m % n);
8 }
9
10 void main()
11 {
12     int m, n, resultado;
13     printf("Para calcular o MDC informe:\n");
14     printf("M:\n");
15     scanf("%d", &m);
16     printf("N:\n");
17     scanf("%d", &n);
18     resultado = mdc(m, n);
19     printf("MDC: %d \n", resultado);
20 }

```

6. Escreva um programa recursivo em linguagem C para converter um número da sua forma decimal para a forma binária. Dica: dividir o número sucessivamente por 2, sendo que o resto da  $i$ -ésima divisão vai ser o dígito  $i$  do número binário (da direita para a esquerda).

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int bin(int k) {

```

```

5     if (k < 2)
6         return k;
7
8     return (10 * bin(k / 2)) + k % 2;
9 }
10
11 void main()
12 {
13     int numero, resultado;
14     printf("Digite um numero:");
15     scanf("%d", &numero);
16
17     resultado = bin(numero);
18
19     printf("Numero binario: %d", resultado);
20 }

```

7. Escreva uma função recursiva em linguagem C para calcular o valor de  $x^n$

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 float potencia(int x, int n) {
5     if (n > 1)
6         return x * potencia(x, n-1);
7     else
8         return x;
9 }
10
11 void main()
12 {
13     int x, n;
14     float resultado;
15     printf("Informe um valor inteiro:\n");
16     scanf("%d", &x);
17     printf("Informe a potencia:\n");
18     scanf("%d", &n);
19     resultado = potencia(x, n);
20     printf("Resultado: %f \n", resultado);
21 }

```

8. Escreva um programa em C recursivo que inverta a ordem dos elementos, números inteiros, de uma lista armazenada em um vetor. Ao final da execução, o conteúdo do primeiro elemento deverá estar no último, o do segundo no penúltimo, e assim por diante. Dica: troque os conteúdos das duas extremidades do vetor e chame uma função recursivamente para fazer o mesmo no subvetor interno.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void inverteVet(int *vet, int ini, int fim) {
5     int aux;
6     aux = vet[ini];

```

```

7   vet[ini] = vet[fim];
8   vet[fim] = aux;
9   if ((fim - ini) > 1)
10      inverteVet(vet, ini+1, fim-1);
11 }
12
13 void main()
14 {
15     int vet[10], i;
16
17     for (i=0; i<10; i++) {
18         printf("Informe um valor inteiro %d:\n", i+1);
19         scanf("%d", &vet[i]);
20     }
21     inverteVet(vet, 0, 9);
22     printf("-----Imprimindo invertido-----\n");
23
24     for (i=0; i<10; i++) {
25         printf("Num %d: %d \n", i+1, vet[i]);
26     }
27 }

```

9. Escreva uma função recursiva para calcular a função de Ackermann  $A(m,n)$ , sendo  $m$  e  $n$  valores inteiros não negativos, dada por:

$$\begin{aligned}
 & n + 1 && \text{se } m = 0 \\
 A(m, n) &= A(m - 1, 1) && \text{se } m > 0 \text{ e } n = 0 \\
 & A(m - 1, A(m, n - 1)) && \text{se } m > 0 \text{ e } n > 0
 \end{aligned}$$

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int ackermann(int m, int n){
5      if (m == 0){
6          return (n+1);
7      }
8      else if ((m > 0) && (n == 0)){
9          return ackermann(m-1, 1);
10     }
11     else if ((m > 0) && (n > 0)){
12         return ackermann(m-1, ackermann(m, n-1));
13     }
14 }
15
16 void main()
17 {
18     int m, n, resultado;
19     printf("Digite o valor de m e n: ");
20     scanf("%d %d", &m, &n);
21     resultado = ackermann(m, n);
22     printf("A(%d,%d) = %d\n", m, n, resultado);
23 }

```

10. Imagine que  $comm(n,k)$  representa o número de diferentes comitês de  $k$  pessoas, que podem ser formados, dadas  $n$  pessoas a partir das quais escolher. Por exemplo,  $comm(4,3) = 4$ , porque dadas quatro pessoas, **A**, **B**, **C** e **D** existem quatro possíveis comitês de três pessoas: **ABC**, **ABD**, **ACD** e **BCD**. Escreva e teste um programa recursivo em C para calcular  $comm(n,k)$  para  $n, k \geq 1$ . Para tal, considere a seguinte identidade:

$$\begin{aligned} comm(n,k) &= n && \text{se } k = 1 \\ comm(n,k) &= 1 && \text{se } k = n \\ comm(n,k) &= comm(n-1, k) + comm(n-1, k-1) && \text{se } 1 < k < n \end{aligned}$$

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int comm(int n, int k){
5     if (k > 1 && k < n){
6         return comm(n-1, k) + comm(n-1, k-1);
7     }
8     else if (k == 1)
9         return n;
10    else if (k == n)
11        return 1;
12 }
13
14 void main()
15 {
16     int n, k, resultado;
17     printf("Digite o valor de n e k: ");
18     scanf("%d %d", &n, &k);
19     resultado = comm(n, k);
20     printf("comm(%d,%d) = %d\n", n, k, resultado);
21 }
```



## Exercícios Resolvidos da Aula 9

1. Escreva um programa que contenha duas variáveis inteiras. Compare seus endereços e exiba o maior endereço.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int num1, num2;
7     if (&num1 > &num2)
8         printf("Endereco da primeira variavel: %d \n", &num1);
9     else
10        printf("Endereco da segunda variavel: %d \n", &num2);
11 }
```

2. Crie um programa que leia números reais em um vetor de tamanho 10. Imprima o endereço de cada posição desse vetor.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     float vetNum[10];
7     int i;
8     for (i=0; i<10; i++)
9         printf("Endereco da posicao %d: %d \n", i+1, &vetNum[i]);
10 }
```

3. Crie um programa que contenha um vetor de inteiros com tamanho 5. Utilizando apenas ponteiros, leia valores e armazene neste vetor e após isso, imprima o dobro de cada valor lido.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
```

```

5 {
6     int vetNum[5], *pNum;
7     int i;
8     for (i=0; i<5; i++) {
9         pNum = &vetNum[i];
10        printf("Informe um numero inteiro: \n");
11        scanf("%d", pNum);
12    }
13
14    for (i=0; i<5; i++) {
15        pNum = &vetNum[i];
16        printf("Dobro do valor: %d \n", *pNum*2);
17    }
18 }

```

4. Elabore um programa que leia um valor do tipo inteiro e, por meio de função, atualize todas as posições de um vetor com o número inteiro lido, depois imprima os valores. Utilize ponteiros para as operações.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int atualiza(int *vet, int num, int tam) {
5     int i, *pnum;
6     for (i=0; i<tam; i++) {
7         pnum = &vet[i];
8         *pnum = num;
9     }
10 }
11
12 void main()
13 {
14     int i, numeros[10], num, *pnumero;
15     printf("Informe o numero: \n");
16     scanf("%d", &num);
17
18     atualiza(numeros, num, 10);
19
20     for (i=0; i<10; i++) {
21         pnumero = &numeros[i];
22         printf("Numero: %d \n", *pnumero);
23     }
24 }

```

5. Faça um programa que receba dois valores inteiros, após receber esses dois valores, uma função deve calcular e retornar para o programa o resultado da soma e da subtração dos valores. Obs.: Apenas uma função deve realizar esta operação, desta forma, faça uso de ponteiros.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void efetuaOperacoes(int valor1, int valor2, int *psoma, int *psubtracao) {
5     *psoma = valor1 + valor2;

```

```

6     *psubtracao = valor1 - valor2;
7 }
8
9 void main()
10 {
11     int num1, num2;
12     int soma, subtracao;
13     printf("Informe o primeiro valor inteiro: \n");
14     scanf("%d", &num1);
15     printf("Informe o segundo valor inteiro: \n");
16     scanf("%d", &num2);
17
18     efetuaOperacoes(num1, num2, &soma, &subtracao);
19
20     printf("Soma: %d \n", soma);
21     printf("Subtracao: %d \n", subtracao);
22 }

```

6. Construa uma função que, recebendo como parâmetros quatro números inteiros, devolva ao módulo que o chamou os dois maiores números dentre os quatro recebidos. Faça um programa que leia tantos conjuntos de quatro valores quantos o usuário deseje e que acione a função para cada conjunto de valores, apresentando a cada vez os dois maiores números. Se preferir, utilize vetor para armazenar o conjunto de valores.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void obtemMajores(int *vet, int *pmaior1, int *pmaior2) {
5     int i, j, aux;
6     for (i=0; i<4; i++) {
7         for (j=i+1; j<4; j++) {
8             if (vet[i] < vet[j]) {
9                 aux = vet[i];
10                vet[i] = vet[j];
11                vet[j] = aux;
12            }
13        }
14    }
15    *pmaior1 = vet[0];
16    *pmaior2 = vet[1];
17 }
18
19 void main()
20 {
21     int vetNum[4], i;
22     int maior1, maior2;
23
24     for (i=0; i<4; i++) {
25         printf("Informe o valor inteiro %d: \n", i+1);
26         scanf("%d", &vetNum[i]);
27     }
28
29     obtemMajores(vetNum, &maior1, &maior2);

```

```

30
31     printf("Os dois maiores valores sao: %d e %d \n", maior1, maior2);
32 }

```

7. Considere um vetor de 10 elementos, contendo valores inteiros. Faça um programa que leia os valores para preencher esse vetor, após a leitura o programa deve invocar uma função que calcule e devolva as frequências absoluta e relativa desses valores no conjunto. (Observação: a frequência absoluta de um valor é o número de vezes que esse valor aparece no conjunto de dados; a frequência relativa é a frequência absoluta dividida pelo número total de dados.). Utilize outros dois vetores para armazenar as frequências relativas e absolutas e ao final do programa, imprima de forma tabulada os números e suas frequências absoluta e relativa.

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void frequencias(int *vet, int tam, int *pAbs, float *pRel) {
5      int i, j, x, aux;
6      //ordenando os dados
7      for (i=0; i<tam; i++) {
8          for (j=i+1; j<tam; j++) {
9              if (vet[i] > vet[j]) {
10                 aux = vet[i];
11                 vet[i] = vet[j];
12                 vet[j] = aux;
13             }
14         }
15     }
16
17     //calculando as frequencias
18     for (i=0; i<tam; i++) {
19         pAbs[i] = 1;
20         for (j=i+1; j<tam; j++) {
21             if (vet[i] == vet[j])
22                 pAbs[i]++;
23             else {
24                 if (pAbs[i] > 1) {
25                     for (x=i+1; x<(i+pAbs[i]); x++) {
26                         pAbs[x] = pAbs[i];
27                         pRel[x] = (float)pAbs[i] / tam;
28                     }
29                 }
30                 break;
31             }
32         }
33         pRel[i] = (float)pAbs[i] / tam;
34         i += pAbs[i]-1;
35     }
36 }
37
38 void main()
39 {
40     int vetNum[10], i, freqAbs[10];

```

```

41     float freqRel[10];
42
43     for (i=0; i<10; i++) {
44         printf("Informe o valor inteiro %d: \n", i+1);
45         scanf("%d", &vetNum[i]);
46     }
47
48     frequencias(vetNum, 10, freqAbs, freqRel);
49
50     printf("Imprimindo os dados\nNumero\tFreq abs\tFreq Rel\n");
51     for (i=0; i<10; i++) {
52         printf("%d\t%d\t\t%f\n", vetNum[i], freqAbs[i], freqRel[i]);
53     }
54 }

```

8. O laboratório de agropecuária da Universidade Federal do Capa Bode tem um termômetro de extrema precisão, utilizado para aferir as temperaturas de uma estufa onde cultivam uma variedade de jaca transgênica, com apenas um caroço do tamanho de uma semente de laranja. O problema é que este termômetro dá os resultados na escala Kelvin (K) e os pesquisadores que atuam perto da estufa são americanos, acostumados com a escala Fahrenheit (F). Você deve criar um programa para pegar uma lista de 24 temperaturas em Kelvin e convertê-las para Fahrenheit. O problema maior é que esses pesquisadores querem que você faça essa conversão e imprima os resultados utilizando ponteiros. Para a conversão, observe as fórmulas a seguir:

$$F = 1.8 \times (K - 273) + 32 \quad (\text{I.1})$$

Em que:

- F = Fahrenheit
- K = Kelvin

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int i;
7     float temperatura[24];
8     float *p_temp;
9
10    //lendo as temperaturas
11    printf("Insira 10 temperaturas em Kelvin: \n");
12    for (i=0; i<=23; i++) {
13        printf("Temperatura %d : ", i+1);
14        scanf("%f", &temperatura[i]);
15    }
16    //atualiza os valores dentro do vetor utilizando ponteiros
17    for (i=0; i<=23; i++) {
18        p_temp = &temperatura[i];
19        *p_temp = 1.8 * temperatura[i] - 459.67; // convertendo temperatura
           para Fahrenheit

```

```

20     }
21
22     printf("Temperaturas convertidas para Fahrenheit: \n");
23     //imprimindo os valores (atualizados) do vetor utilizando ponteiros
24     for (i=0; i<=23; i++) {
25         p_temp = &temperatura[i];
26         printf("Temperatura %d : %.2f \n", i+1, *p_temp);
27     }
28 }

```

9. A Google está desenvolvendo um novo sistema operacional para máquinas de venda de bolinhas de borracha de R\$1,00, mas precisa realizar testes no Gerenciador de Memória desse novo sistema. Você foi contratado para fazer um programa para verificar se o gerenciador de memória está funcionando corretamente. Seu programa deverá ler 3 números inteiros, 3 números decimais, 3 letras, armazená-las em variáveis, e depois, através de ponteiros, trocar os seus valores, substituindo todos os números inteiros pelo número 2014, os decimais por 9.99, e as letras por 'Y'. Depois da substituição, o programa deverá exibir o valor das variáveis já devidamente atualizados.

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void main()
5  {
6      int num1, num2, num3;
7      int *p_num;
8      float dec1, dec2, dec3;
9      float *p_dec;
10     char letra1, letra2, letra3;
11     char *p_letra;
12
13     // lendo os dados necessarios
14     printf("Digite um numero inteiro: ");
15     scanf("%d", &num1);
16     printf("Digite outro numero inteiro: ");
17     scanf("%d", &num2);
18     printf("Digite outro numero inteiro: ");
19     scanf("%d", &num3);
20
21     printf("Digite um numero decimal: ");
22     scanf("%f", &dec1);
23     printf("Digite outro numero decimal: ");
24     scanf("%f", &dec2);
25     printf("Digite outro numero decimal: ");
26     scanf("%f", &dec3);
27
28     printf("Digite uma letra: ");
29     scanf(" %c", &letra1);
30     printf("Digite outra letra: ");
31     scanf(" %c", &letra2);
32     printf("Digite outra letra: ");
33     scanf(" %c", &letra3);
34

```

```

35 //imprimindo dados que foram lidos
36 printf("Os numeros informados foram: \n");
37 printf("Inteiros: %d, %d e %d \n", num1, num2, num3);
38 printf("Decimais: %.2f, %.2f e %.2f \n", dec1, dec2, dec3);
39 printf("Letras: %c, %c e %c \n", letra1, letra2, letra3);
40 printf("\n");
41
42 //troca os valores das variaveis atraves de ponteiros
43 p_num = &num1;
44 *p_num = 2014;
45 p_num = &num2;
46 *p_num = 2014;
47 p_num = &num3;
48 *p_num = 2014;
49
50 p_dec = &dec1;
51 *p_dec = 9.99;
52 p_dec = &dec2;
53 *p_dec = 9.99;
54 p_dec = &dec3;
55 *p_dec = 9.99;
56
57 p_letra = &letra1;
58 *p_letra = 'Y';
59 p_letra = &letra2;
60 *p_letra = 'Y';
61 p_letra = &letra3;
62 *p_letra = 'Y';
63
64 //imprime as variaveis com os novos valores
65 printf("Valores modificados: \n");
66 printf("Primeiro numero inteiro: %d \n", num1);
67 printf("Segundo numero inteiro: %d \n", num2);
68 printf("Terceiro numero inteiro: %d \n", num3);
69
70 printf("Primeiro numero decimal: %.2f \n", dec1);
71 printf("Segundo numero decimal: %.2f \n", dec2);
72 printf("Terceiro numero decimal: %.2f \n", dec3);
73
74 printf("Primeira letra: %c \n", letra1);
75 printf("Segunda letra: %c \n", letra2);
76 printf("Terceira letra: %c \n", letra3);
77 }

```

10. O departamento comercial da Batatinha S/A necessita atualizar os valores de seus produtos no seu catálogo de vendas. O presidente ordenou um reajuste de 4.78% para todos os itens. São 15 itens no catálogo. Sua tarefa é elaborar um programa que leia o valor atual dos produtos e armazene em um vetor, e após isso efetue o reajuste no valor dos produtos. O reajuste (acesso ao vetor) deverá ser feito utilizando ponteiros. Imprima na tela o valor reajustado, usando também ponteiros.

```

1 #include <stdio.h>
2 #include <stdlib.h>

```

```
3
4 void main()
5 {
6     int i;
7     float catalogo[15], *p_cat;
8     //coletando valor dos produtos
9     printf("Informe os valor dos produtos: \n");
10    for(i=0; i<=14; i++) {
11        printf("Informe o valor do item %d do catalogo: ", i+1);
12        scanf("%f", &catalogo[i]);
13    }
14
15    //atualizando catalogo de precos atraves do ponteiro
16    for(i=0; i<=14; i++) {
17        p_cat = &catalogo[i];
18        *p_cat = *p_cat+(*p_cat * 0.0478);
19        //imprimindo catalogo(atualizados)
20        printf("Preco reajustado do item %d : %.2f \n", i+1, *p_cat );
21    }
22 }
```

## Exercícios Resolvidos da Aula 10

1. Escreva um programa que mostre o tamanho em byte que cada tipo de dados ocupa na memória: **char**, **int**, **float**, **double**.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     printf("Tamanho caracter...: %d byte(s)\n", sizeof(char));
7     printf("Tamanho inteiro...: %d byte(s)\n", sizeof(int));
8     printf("Tamanho real.....: %d byte(s)\n", sizeof(float));
9     printf("Tamanho real longo.: %d byte(s)\n", sizeof(double));
10 }
```

2. Elabore um programa que leia do usuário o tamanho de um vetor a ser lido. Em seguida, faça a alocação dinâmica desse vetor. Por fim, leia o vetor do usuário e o imprima.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int i, tam, *pVetor;
7     printf("Informe o tamanho desejado:\n");
8     scanf("%d", &tam);
9
10    //alocando memoria do tamanho requisitado
11    pVetor = (int*) malloc(tam * sizeof(int));
12
13    //lendo valores e armazenando no vetor
14    for (i=0; i<tam; i++) {
15        printf("Informe o valor inteiro %d:\n", i+1);
16        scanf("%d", &pVetor[i]);
17    }
18
19    //imprimindo os valores
20    for (i=0; i<tam; i++) {
```

```

21     printf("Valor: %d\n", pVetor[i]);
22     }
23     free(pVetor);
24 }

```

3. Faça um programa que leia um valor inteiro  $N$  não negativo. Se o valor de  $N$  for inválido, o usuário deverá digitar outro até que ele seja válido (ou seja, positivo). Em seguida, leia um vetor  $V$  contendo  $N$  posições de inteiros, em que cada valor deverá ser maior ou igual a 2. Esse vetor deverá ser alocado dinamicamente.

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void main()
5  {
6      int i, N, *V;
7
8      do {
9          printf("Informe um valor N nao negativo:\n");
10         scanf("%d", &N);
11         if (N <= 0)
12             printf("Valor invalido!\n");
13         } while (N <= 0);
14
15         //alocando memoria do tamanho requisitado
16         V = (int*) malloc(N * sizeof(int));
17
18         //lendo valores e armazenando no vetor
19         for (i=0; i<N; i++) {
20             do {
21                 printf("Informe o valor inteiro >= 2:\n");
22                 scanf("%d", &V[i]);
23                 if (V[i] < 2)
24                     printf("Valor invalido!\n");
25             } while (V[i] < 2);
26         }
27
28         //imprimindo os valores
29         for (i=0; i<N; i++) {
30             printf("Valor: %d\n", V[i]);
31         }
32         free(V);
33     }

```

4. Faça uma função que retorne o ponteiro para um vetor de  $N$  elementos inteiros alocados dinamicamente. O vetor deve ser preenchido com valores de  $0$  a  $N-1$ .

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int *alocaVet(int tam) {
5      int i, *vet;
6      //alocando memoria do tamanho requisitado
7      vet = (int*) malloc(tam * sizeof(int));

```

```
8
9     for (i=0; i<tam; i++) {
10         vet[i] = i;
11     }
12     return vet;
13 }
14
15 void main()
16 {
17     int i, N, *V;
18
19     printf("Informe um valor N:\n");
20     scanf("%d", &N);
21
22     V = alocaVet(N);
23
24     //imprimindo os valores
25     for (i=0; i<N; i++) {
26         printf("Valor: %d\n", V[i]);
27     }
28     free(V);
29 }
```

5. Escreva uma função que receba um valor inteiro positivo N por parâmetro e retorne o ponteiro para um vetor de tamanho N alocado dinamicamente. Se N for negativo ou igual a zero, um ponteiro nulo deverá ser retornado.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int *alocaVet(int tam) {
5     int i, *vet;
6
7     if (tam > 0) {
8         //alocando memoria do tamanho requisitado
9         vet = (int*) malloc(tam * sizeof(int));
10        return vet;
11    }
12    else
13        return NULL;
14 }
15
16 void main()
17 {
18     int i, N, *V;
19
20     printf("Informe um valor N:\n");
21     scanf("%d", &N);
22
23     V = alocaVet(N);
24
25     if (V != NULL) {
26         printf("Alocado corretamente!\n");
27         free(V);
28     }
```

```

28     }
29     else
30         printf("Erro na alocação!\n");
31 }

```

6. Crie uma função que receba um texto e retorne o ponteiro para esse texto invertido.

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  char *inverte(char *texto) {
6      int i, j=0;
7      char *texto2 = malloc(strlen(texto) * sizeof(char));
8      //invertendo o texto
9      for (i=strlen(texto)-1; i>=0; i--) {
10         texto2[j] = texto[i];
11         j++;
12     }
13     texto2[j] = '\0';
14     return texto2;
15 }
16
17 void main()
18 {
19     char str[20];
20     char *str2;
21     printf("Informe um texto com tamanho ate 20:\n");
22     fgets(str, 20, stdin);
23     //aciona a funcao para inverter
24     str2 = inverte(str);
25     //imprimindo invertido
26     printf("Invertido = %s \n", str2);
27     //liberando a memoria
28     free(str2);
29 }

```

7. Escreva uma função que receba como parâmetro dois vetores, **A** e **B**, de tamanho **N** cada. A função deve retornar o ponteiro para um vetor **C** de tamanho **N** alocado dinamicamente, em que  $C[i] = A[i] + B[i]$ .

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int *soma(int *vet1, int *vet2, int tam) {
5      int i;
6      //alocando memoria
7      int *vet3 = (int*)malloc(tam * sizeof(int));
8      //somando os vetores
9      for (i=0; i<tam; i++) {
10         vet3[i] = vet1[i] + vet2[i];
11     }
12     return vet3;

```

```

13 }
14
15 void main()
16 {
17     int tam, i, *pvet1, *pvet2, *pvet3;
18     printf("Informe o tamanho desejado:\n");
19     scanf("%d", &tam);
20     //alocando memoria
21     pvet1 = (int*)malloc(tam * sizeof(int));
22     pvet2 = (int*)malloc(tam * sizeof(int));
23     //lendo os dados
24     for (i=0; i<tam; i++) {
25         printf("Informe o numero %d do vetor 1:\n", i+1);
26         scanf("%d", &pvet1[i]);
27     }
28     for (i=0; i<tam; i++) {
29         printf("Informe o numero %d do vetor 2:\n", i+1);
30         scanf("%d", &pvet2[i]);
31     }
32     //aciona a funcao
33     pvet3 = soma(pvet1, pvet2, tam);
34
35     for (i=0; i<tam; i++) {
36         printf("Numero: %d\n", pvet3[i]);
37     }
38     //liberando a memoria
39     free(pvet1);
40     free(pvet2);
41     free(pvet3);
42 }

```

8. Escreva uma função que receba como parâmetro dois vetores, **A** e **B**, de tamanho **N** cada. A função deve retornar o ponteiro para um vetor **C** de tamanho  $N \times 2$  alocado dinamicamente, em que  $|C| = |A| + |B|$ , ou seja, a união dos dois conjuntos irão formar o conjunto **C**, ou vetor **C**.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int *junta(int *vet1, int *vet2, int tam) {
5     int i;
6     //alocando memoria
7     int *vet3 = (int*)malloc(tam*2 * sizeof(int));
8     //juntando os vetores
9     for (i=0; i<tam; i++) {
10         vet3[i] = vet1[i];
11     }
12     for (i=tam; i<tam*2; i++) {
13         vet3[i] = vet2[i-tam];
14     }
15     return vet3;
16 }
17
18 void main()

```

```

19 {
20     int tam, i, *pvet1, *pvet2, *pvet3;
21     printf("Informe o tamanho desejado:\n");
22     scanf("%d", &tam);
23     //alocando memoria
24     pvet1 = (int*)malloc(tam * sizeof(int));
25     pvet2 = (int*)malloc(tam * sizeof(int));
26     //lendo os dados
27     for (i=0; i<tam; i++) {
28         printf("Informe o numero %d do vetor 1:\n", i+1);
29         scanf("%d", &pvet1[i]);
30     }
31     for (i=0; i<tam; i++) {
32         printf("Informe o numero %d do vetor 2:\n", i+1);
33         scanf("%d", &pvet2[i]);
34     }
35     //aciona a funcao
36     pvet3 = junta(pvet1, pvet2, tam);
37
38     for (i=0; i<tam*2; i++) {
39         printf("Numero: %d\n", pvet3[i]);
40     }
41     //liberando a memoria
42     free(pvet1);
43     free(pvet2);
44     free(pvet3);
45 }

```

9. Escreva um programa que aloque dinamicamente uma matriz de inteiros. As dimensões da matriz deverão ser lidas do usuário. Em seguida, escreva uma função que receba um valor e retorne **1**, caso o valor esteja na matriz, ou retorne **0**, no caso contrário.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int busca(int valor, int **matriz, int tamRol, int tamCol) {
5     int i, j;
6     for (i=0; i<tamRol; i++) {
7         for (j=0; j<tamCol; j++) {
8             if (valor == matriz[i][j])
9                 return 1;
10        }
11    }
12    return 0;
13 }
14
15 void main()
16 {
17     int tamRol, tamCol, i, j, **pMatriz;
18     int valorConsulta, resultado;
19     printf("Informe o numero de linhas da matriz:\n");
20     scanf("%d", &tamRol);
21     printf("Informe o numero de colunas da matriz:\n");

```

```

22     scanf("%d", &tamCol);
23
24     //alocando memoria e lendo valores
25     pMatriz = (int*)malloc(tamRol * sizeof(int));
26     for (i=0; i<tamRol; i++) {
27         pMatriz[i] = (int*)malloc(tamCol * sizeof(int));
28         for (j=0; j<tamCol; j++) {
29             printf("Informe um valor inteiro para a matriz: (%d,%d)\n", i,
30                 j);
31             scanf("%d", &pMatriz[i][j]);
32         }
33     }
34     printf("Informe um valor inteiro para consulta:\n");
35     scanf("%d", &valorConsulta);
36
37     //consultando
38     resultado = busca(valorConsulta, pMatriz, tamRol, tamCol);
39     if (resultado == 1)
40         printf("Encontrou!\n");
41     else
42         printf("Nao encontrou!\n");
43
44     //liberando a memoria
45     for (i=0; i<tamRol; i++) {
46         free(pMatriz[i]);
47     }
48     free(pMatriz);

```

10. Escreva um programa que leia um inteiro **N** e crie uma matriz alocada dinamicamente contendo **N** linhas e **N** colunas. Essa matriz deve conter o valor **0** na diagonal principal, o valor **1** nos elementos acima da diagonal principal e o valor **-1** nos elementos abaixo da diagonal principal. Veja a figura 29 para entender melhor o preenchimento da matriz.

0	1	1	1
-1	0	1	1
-1	-1	0	1
-1	-1	-1	0

Diagonal principal

Figura 29 – Formato de preenchimento da matriz para o exercício

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int N, i, j, **pMatriz;
7     printf("Informe o numero N:\n");
8     scanf("%d", &N);

```

```
9
10 //alocando memoria
11 pMatriz = (int*)malloc(N * sizeof(int));
12 for (i=0; i<N; i++) {
13     pMatriz[i] = (int*)malloc(N * sizeof(int));
14     for (j=0; j<N; j++) {
15         if (i == j) {
16             //esta na diagonal principal
17             pMatriz[i][j] = 0;
18         }
19         else if (i < j) {
20             pMatriz[i][j] = 1;
21         }
22         else {
23             pMatriz[i][j] = -1;
24         }
25     }
26 }
27 //imprimindo
28 for (i=0; i<N; i++) {
29     for (j=0; j<N; j++) {
30         if (j < (N-1))
31             printf("%d\t", pMatriz[i][j]);
32         else
33             printf("%d\n", pMatriz[i][j]);
34     }
35 }
36
37 //liberando a memoria
38 for (i=0; i<N; i++) {
39     free(pMatriz[i]);
40 }
41 free(pMatriz);
42 }
```

# Referências

ALBANO, R. S.; ALBANO, S. G. *Programação Em Linguagem C*. 1. ed. Rio de Janeiro: Ciência Moderna, 2010. ISBN 8573939494. Citado 3 vezes nas páginas [31](#), [57](#) e [78](#).

ANICHE, M. *Introdução à programação em C: Os primeiros passos de um desenvolvedor*. 1. ed. São Paulo: Casa do Código, 2015. ISBN 9788555190889. Citado 6 vezes nas páginas [30](#), [56](#), [62](#), [78](#), [112](#) e [139](#).

ASCENCIO, A. F. G.; CAMPOS, E. A. V. de. *Fundamentos da programação de computadores: Algoritmos, Pascal e C/C++*. 1. ed. São Paulo: Pearson, 2002. ISBN 8567918362. Citado 2 vezes nas páginas [31](#) e [89](#).

BACKES, A. *Linguagem C: completa e descomplicada*. 1. ed. Rio de Janeiro: Elsevier, 2013. ISBN 9788535268553. Citado 18 vezes nas páginas [31](#), [50](#), [63](#), [78](#), [81](#), [82](#), [89](#), [98](#), [105](#), [109](#), [144](#), [145](#), [148](#), [152](#), [153](#), [155](#), [160](#) e [163](#).

BACKES, A. *Estrutura de dados descomplicada: em linguagem C*. 1. ed. Rio de Janeiro: Elsevier, 2016. ISBN 9788535285239. Nenhuma citação no texto.

BÄCKMAN, K. *Structured Programming with C++*. London: bookboon.com, 2012. ISBN 9788740300994. Citado na página [50](#).

DAMAS, L. *Linguagem C*. 10. ed. Rio de Janeiro: LTC, 2007. ISBN 85-216-1519-1. Citado 4 vezes nas páginas [10](#), [11](#), [31](#) e [36](#).

EDELWEISS, N.; LIVI, M. A. C. *Algoritmos e programação com exemplos em Pascal e C*. Porto Alegre: Bookman, 2014. ISBN 9788582601891. Citado 10 vezes nas páginas [33](#), [89](#), [98](#), [99](#), [128](#), [134](#), [139](#), [148](#), [152](#) e [153](#).

FEOFILOFF, P. *Algoritmos em Linguagem C*. 1. ed. Rio de Janeiro: Elsevier, 2008. ISBN 8535232494. Citado 2 vezes nas páginas [128](#) e [140](#).

GOOKIN, D. *Começando a Programar em C Para Leigos*. 1. ed. Rio de Janeiro: Alta Books, 2016. ISBN 8576089750. Citado 2 vezes nas páginas [57](#) e [89](#).

HASKINS, D. *C Programming in Linux*. 2. ed. London: bookboon.com, 2013. ISBN 9788740305432. Citado 2 vezes nas páginas [30](#) e [50](#).

KERNIGHAN, B. W.; RITCHIE, D. M. *The C programming language*. 2. ed. New Jersey: Prentice Hall Software Series, 1988. ISBN 0131103709. Citado 4 vezes nas páginas [50](#), [52](#), [56](#) e [62](#).

- LAUREANO, M. *Programando em C para Linux, Unix e Windows*. Rio de Janeiro: Brasport, 2005. ISBN 85-7452-233-3. Citado 9 vezes nas páginas [12](#), [13](#), [16](#), [17](#), [18](#), [41](#), [112](#), [121](#) e [140](#).
- LOPES, A.; GARCIA, G. *Introdução à programação: 500 algoritmos resolvidos*. Rio de Janeiro: Elsevier, 2002. ISBN 85-352-1019-9. Citado 20 vezes nas páginas [27](#), [30](#), [31](#), [32](#), [36](#), [38](#), [39](#), [42](#), [45](#), [51](#), [58](#), [60](#), [61](#), [64](#), [70](#), [81](#), [84](#), [88](#), [94](#) e [124](#).
- MANZANO, J. A. N. G. *Linguagem C. Acompanhada de Uma Xícara de Café*. 1. ed. São Paulo: Érica, 2015. ISBN 8536514620. Citado 2 vezes nas páginas [30](#) e [50](#).
- MIZRAHI, V. V. *Treinamento em Linguagem C. Curso Completo em 1 Volume*. 2. ed. São Paulo: Pearson, 2008. ISBN 8576051915. Citado 3 vezes nas páginas [41](#), [63](#) e [89](#).
- PAES, R. de B. *Introdução à Programação com a Linguagem C*. 1. ed. Rio de Janeiro: Novatec, 2016. ISBN 8575224859. Citado 2 vezes nas páginas [15](#) e [31](#).
- SCHILD, H. *C, completo e total*. 3. ed. São Paulo: Makron Books, 1996. ISBN 85-346-0595-5. Citado 11 vezes nas páginas [15](#), [21](#), [38](#), [56](#), [57](#), [62](#), [63](#), [78](#), [112](#), [113](#) e [115](#).
- SILVA, R. L.; OLIVEIRA, A. M. *Algoritmos em C*. 1. ed. Juiz de Fora: Clube de Autores, 2014. ISBN 9788591769704. Citado 2 vezes nas páginas [50](#) e [89](#).
- SZWARCFITER, J. L.; MARKENZON, L. *Estruturas de Dados e Seus Algoritmos*. 3. ed. Rio de Janeiro: LTC, 2015. ISBN 9788521629948. Citado na página [128](#).
- TENENBAUM, A. M.; LANGSAM, Y.; AUGENSTEIN, M. J. *Estruturas de dados usando C*. 1. ed. São Paulo: MAKRON Books, 1995. ISBN 8534603480. Citado 4 vezes nas páginas [128](#), [129](#), [130](#) e [134](#).

Esta página foi deixada em branco de propósito.