Características generales del lenguaje de programación C

Autor: Moreno Madrona, Natividad (Ingeniera Técnica en Informática de Gestión e Ingeniera Técnica en Informática de Sistemas, Profesor de Enseñanza Secundaria).
Público: Ingeniería en Informática. **Materia:** Programación en lenguajes estructurados. **Idioma:** Español.

Título: Características generales del lenguaje de programación C.

Resumen
C fue originalmente desarrollado como lenguaje para implementaciones de S.O. Sin embargo, actualmente se ha construido gran cantidad de productos sw, aplicaciones empotradas y sw de sistemas. Permite al programador "acercarse a la máquina" al suministrar posibilidades similares a las del lenguaje ensamblador. El lenguaje no tiene ninguna posibilidad de realizar asignación de memoria aparte de las definiciones estáticas y el manejo de pilas para las variables locales de las funciones. C posee un alto grado de portabilidad (movilidad). Debido a que los tipos de datos y estructuras de control que ofrece son manejados por la mayoría de computadoras.
Palabras clave: lenguaje de programación, tipos de datos.

Title: General characteristics of the programming language C.

Abstract
C was originally developed as a language for implementations S.O. However, it has now been built sw lot of products, embedded applications and sw systems. It allows the programmer to " approach the machine " by supplying similar to assembly language possibilities. The language has no chance of making memory allocation beyond static definitions and handling batteries for local variables of functions. C has a high degree of portability (mobility). Because the data types and control structures offered are handled by most computers.
Keywords: programming language, data types.

Recibido 2016-03-30; Aceptado 2016-04-01; Publicado 2016-04-25; Código PD: 070100

1. Introducción. Características generales.

C fue originalmente desarrollado como lenguaje para implementaciones de S.O. Sin embargo, actualmente se ha construido gran cantidad de productos sw, aplicaciones empotradas y sw de sistemas. Permite al programador "acercarse a la máquina" al suministrar posibilidades similares a las del lenguaje ensamblador.

C no contiene operaciones para trabajar directamente con objetos compuestos, tales como cadenas de caracteres, conjuntos, listas o arrays, considerados como un todo. El lenguaje no tiene ninguna posibilidad de realizar asignación de memoria aparte de las definiciones estáticas y el manejo de pilas para las variables locales de las funciones. Finalmente, C no cuenta con operaciones de E/S: no existen proposiciones READ o WRITE, ni métodos propios para el acceso a archivos.

C posee un alto grado de portabilidad (movilidad). Debido a que los tipos de datos y estructuras de control que ofrece son manejados por la mayoría de computadoras. A pesar de esto, el lenguaje es independiente de cualquier arquitectura de máquina.

Muchas de las ideas fundamentales de C provienen de un lenguaje mucho más antiguo: el lenguaje BCPL. A diferencia de C, BCPL es un lenguaje "si tipos". En C los objetos fundamentales son caracteres, enteros de varios tamaños y números en punto flotante, además de una jerarquía de tipos de datos derivados.

2. Elementos del lenguaje.

2.1. Comentarios.

Se utilizan para documentar el código fuente de un programa. En C, un comentario es cualquier cadena delimitada por los caracteres /* y */, y puede ocupar mas de una línea.

2.2. Variables.

Son los objetos básicos que se manipulan en un programa.

2.2.1. Nombres de variables.

Los nombres se construyen con letras y dígitos. El primer carácter debe ser una letra. Se diferencia entre mayúsculas y minúsculas. El carácter subrayado "_" se considera igual que una letra. Las palabras claves como if, else, int, float, etc. están reservadas: no se pueden usar como nombres de variables.

2.2.2. Tipos de datos.

Los tipos de datos existentes en C son:

- char -> un byte (octeto) capaz de contener un carácter del juego de caracteres de la instalación local.
- int -> un entero, normalmente del tamaño de los enteros de la instalación local.
- float -> un número en punto flotante en precisión normal.
- double -> un número en punto flotante en doble precisión.

Además, hay una serie de codificadores que se aplican a los enteros:

- short y long -> Dependen del compilador. Siempre que long>entero>short.
- signed y unsigned -> Se aplican también a datos de tipo char. Los números unsigned son siempre positivos o 0.
- long puede aplicarse también al tipo double y especifica punto flotante de precisión extendida.

Además C permite la utilización de punteros, que veremos más adelante.

2.2.3. Declaraciones.

Todas las variables deben ser declaradas antes de utilizarlas. Una declaración especifica un tipo, y le sigue una lista de una o más variables de ese tipo. Las variables se pueden inicializar en su declaración.

2.2.4. Conversión de tipos.

Por regla general, a todo operador aritmético se le aplica la siguiente secuencia de reglas de conversión:

- Char y short se convierten en int, y float en double.
- Si algún operando es de tipo double, el otro se convierte en double y el resultado es double.
- Si no se aplica la regla anterior y algún operando es del tipo long, el otro será también long y el resultado es long.
- Si no se aplica la regla anterior y algún operando es unsigned, el otro operando y el resultado son unsigned.
- Si no se puede aplicar la regla anterior, los operandos deben ser int y el resultado será int.

Las conversiones también tienen lugar en asignaciones; el valor de la parte derecha se convierte al tipo de la izquierda, que es el tipo del resultado. Los bits de orden superior de exceso serán sencillamente eliminados.

Por último se puede forzar la conversión mediante una operación denominada cast:

(nombre_de_tipo_nuevo) expresión

2.3. Constantes.

Una constante es un valor que no varía a lo largo del proceso de ejecución del programa. Una expresión constante es una expresión formada únicamente por constantes. Estas expresiones se evalúan en tiempo de compilación.

2.4. Operadores.

2.4.1. Operadores aritméticos.

Binarios: -, +. *, /, % (no puede usarse con float o double)
Unitarios: -

2.4.2. Operadores relacionales y lógicos.

Relacionales: >, >=, <, <=
Igualdad: = =, !=
Conectivos lógicos: &&, || (evaluadas de izquierda a derecha)
Negación: !

2.4.3. Operadores de incremento y decremento.

Prefijos: ++, -- (incrementan/decrementan la variable antes de utilizar su valor).
Sufijos: ++, -- (incrementan/decrementan la variable después de utilizar su valor).

2.4.4. Operadores lógicos y de manejo de bits.

- Son: & (Y lógico), | (O lógico), ^ (O exclusivo lógico), << (desplazamiento a la izquierda), >> (desplazamiento a la derecha), (complemento a 1).
- No se pueden aplicar a float o double.

2.4.5. Operadores y expresiones de asignación.

Las expresiones como i=i+2 en las que el miembro izquierdo se repite en el derecho pueden escribirse utilizando un operador de asignación como +=, de la forma i+=2. La mayor parte de los operadores binarios poseen su correspondiente operador de asignación.

2.4.6. Expresiones condicionales.

En la expresión e1?e2:e3 primero se evalúa e1. Si no es cierto, se evalúa e2, y éste es el valor de la expresión condicional. De lo contrario se evalúa e3 y éste será su valor.

2.4.7. Operadores de punteros.

Un puntero es una variable que contiene la dirección en un objeto, por tanto, se puede acceder al objeto "indirectamente" a través de él. El operador unitario & devuelve la dirección del objeto al que acompaña. El operador unitario * toma su operando como una dirección y accede a esa dirección para obtener su contenido.

2.5. Funciones.

Las funciones dividen grandes trabajos de computación en partes más pequeñas y permiten aprovechar el trabajo realizado por otras personas.

Una función tiene la forma:

```
Tipo_del_valor_devuelto   nombre_función (declaración de argumentos)
{
        Declaraciones y proposiciones si existen
        [return [expresión];]
}
```

Si la función devuelve un entero, el tipo_del_valor_devuelto puede omitirse. La sentencia return es el mecanismo para devolver un valor al llamador. Puede no ir seguido de ninguna expresión o no aparecer.

2.6. Macros.

Una definición de la forma

```
#define SI 1
```

Indica que debe efectuarse un remplazamiento de un nombre (SI) por una cadena de caracteres (1). El "ámbito" de un nombre definido mediante #define se entiende desde el punto de su definición hasta el final del archivo fuente.

2.7. Control de Flujo.

2.7.1. Proposiciones y bloques.

Una proposición es una expresión como x=0 ó i++ o printf(…) seguida por punto y coma, con las llaves { y } se agrupan declaraciones y sentencias en una proposición compuesta o bloque, y son sintácticamente equivalentes a una proposición simple. Nunca se pone ; después de la llave } de cierre de bloque.

2.7.2. If-Else.

Sirve para tomar decisiones:

```
If (expresión)
        Proposición 1
[else
        Proposición 2]
```

Si *expresión* es cierta se ejecuta *proposición 1*, sino se ejecuta *proposición 2* (*else* es opcional).

2.7.3. Switch.

Es una herramienta especial para decisiones múltiples que comprueba si una expresión iguala uno entre varios valores constantes, y se bifurca a partir de ellos:

```
Switch (expresión)
{
  Case valor_1:
        Proposición 1;
        Break;
  Case valor_2:
        Proposición 2;
        Break;
  . . .

  Default:
```

```
        Proposición;
        Break;
}
```

2.7.4. Iteraciones: while y for.

En:

```
While(expresión)
        Proposición
```

Se evalúa *expresión*. Si es cierta, se ejecuta la *proposición* y se vuelve a evaluar la *expresión.* El ciclo continúa hasta que *expresión* sea falsa.

La proposición for:

```
for(expr1; expr2; expr3)
        proposición
```

Es equivalente a:

```
expr1;
while (expr2)
{
        proposición;
        expr3;
}
```

2.7.5. Iteraciones: do-while.

La sintaxis es:

```
do
        proposición
while (expresión);
```

Primero se ejecuta *proposición* y a continuación se evalúa *expresión*. En caso de ser cierta, se vuelve a ejecutar *proposición* y así sucesivamente hasta que *expresión* deja de ser cierta.

2.8. Punteros y arrays.

La declaración

```
int a[10];
```

define un array de tamaño 10, es decir, un bloque de 10 objetos consecutivos denominador a[0], a[1], … , a[9]. La notación a[i] significa el elemento del array "a" que se encuentra a i posiciones del comienzo. Si pa es un puntero a entero, entonces pa=&a[0] hace que pa apunte al elemento cero de a; es decir, pa contiene la dirección de a[0]. Ahora x=*pa copiará en x el contenido de a[0].

2.9. Estructuras.

La palabra clave struct introduce la declaración de una estructura, que no es más que una lista de declaraciones encerrada entre llaves. Los elementos de una estructura se denominan miembros, y se accede a esos miembros mediante

los operadores "." O "->", dependiendo de si accedemos mediante una variable del tipo estructura o mediante un puntero a dicha estructura. La sintaxis de la estructura sería:

```
struct [nombre_estructura]{
        . . .
}x, y, z;          //x, y, z son variables de tipo estructura
```

3. Estructura de un programa.

```
/* CABECERA DEL PROGRAMA*/
Directivas del preprocesador
Declaraciones de variables globales

/* Declaraciones del tipo de dato devuelto por cada función*/
Tipo función1 (tipo, tipo,…);
. . .
Tipo funciónN (tipo, tipo,…);

/* Definición de cada una de las funciones*/
función1()
{
 . . .;
}
. . .
funciónN()
{
 . . .;
}
/* función principal MAIN */
main ()
{
 . . . ;
}
```

3.1. Argumentos de la función main.

Cuando se invoca al main al comienzo de la ejecución, se llama con dos argumentos:

- argc: nº de argumentos en la línea de comandos con los que se invocó al programa.
- argv: puntero a un array de cadenas de caracteres que contienen los argumentos. argv[0] es el nombre con el que se invocó al programa.

3.2. Ficheros cabecera y prototipo de funciones.

Un programa C puede constar de varios ficheros de código fuente. Para evitar tener que declarar las funciones creadas en cada uno de los ficheros fuente, se introduce la declaración o prototipo de la función en un fichero cabecera, de forma que basta realizar un #include de dicho fichero para tener todas las declaraciones.

4. Funciones de Librería y usuario.

Una librería no es más que un archivo donde están agrupados varios módulos o funciones, de tal forma que el enlazador entiende su formato.

Todo archivo fuente que utilice funciones de la biblioteca deberá contener una directiva #include al principio para incluir el fichero cabecera. El nombre del fichero irá entre comillas o entre ángulos si el fichero no se encuentra en el mismo directorio.

Bibliografía

- Dennis M. Ritchie (Enero de 1993). «The Development of the C Language».
- Ritchie (1993)
- http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=57853
- INTERNATIONAL STANDARD ©ISO/IEC ISO/IEC 9899:201x - WG14 N1570 Committee Draft — April 12, 2011