View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Repositorio Institucional de la Universidad Autónoma del Estado de México

UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MÉXICO

FACULTAD DE INGENIERÍA

INGENIERÍA EN COMPUTACIÓN

UNIDAD DE APRENDIZAJE:

ESTRUCTURAS DE DATOS

TEMA:

PILAS, COLAS, LISTAS, ÁRBOLES Y GRAFOS

MATERIAL DIDÁCTICO

PROBLEMARIO PARA PRÁCTICA DE ESTRUCTURAS DE DATOS

ELABORADO POR:

DRA. MIREYA SALGADO GALLEGOS

MAYO 2019

ÍNDICE

PRESENTACIÓN

El programa de Estructuras de Datos tiene por objetivo que el alumno identifique las herramientas teóricas fundamentales para la representación y manipulación de información en la computadora, haciendo énfasis en el tipo de datos dinámicos con base en éste, el programa está conformado en la actualidad de 4 unidades de competencia:

1. Reconocer y manejar las variables dinámicas
2. Aplicar las principales estructuras de datos lineales.
3. Aplicar la estructura de datos árbol.
4. Aplicar la estructura de datos grafo

La forma de impartición de esta unidad de aprendizaje se basa en dos partes, la parte teórica y la parte práctica para cumplir con el número de 3 y 2 horas respectivamente. Este material está enfocado a aplicar la parte teórica con la finalidad de que a los alumnos les queden claros los conceptos de teoría específicamente en la implementación de ésta.

Basado en lo anterior, este material didáctico está orientado principalmente a los alumnos del segundo semestre y a docentes de la licenciatura de Ingeniería en Computación, con la finalidad de apoyar en el desarrollo de habilidades de programación, basadas en la interpretación, razonamiento, análisis y aplicación de la teoría de estructuras de datos dinámicas.

Este material está enfocado únicamente a la realización de prácticas basadas en la parte teórica de los temas que aborda la unidad de aprendizaje, siendo éste un complemento al material denominado "*Actividades para teoría de estructuras de datos*" el cual consta de dos versiones.

Cabe mencionar que se omite toda teoría debido a que es sólo para la aplicación de ésta, ya contemplada en los apuntes de la materia de estructuras de datos.

El material presentado es una recopilación de prácticas (programas) que el alumno irá realizando con base en la teoría expuesta previamente.

Por motivos de la naturaleza del material en el apartado de la solución de los problemas sólo se anexan algunas prácticas resueltas debido a que las actividades son todas diferentes.

Al final de este documento se anexa una bibliografía con la finalidad de que el alumno pueda, si así lo requiere, consultar algún material para su apoyo así como también un apartado de anexos en el cual se presentan las implementaciones de las operaciones de pila, cola, listas, árboles y grafos ya que éstas pueden apoyar en la solución de los prácticas de estas estructuras de datos.

Los conocimientos previos requeridos para este material son los temas de pila, cola, lista en sus diferentes modalidades, árboles y grafos.

GUÍA DE USO

La presente guía de uso pretende orientar la aplicación de este material, describiendo las partes de éste y ejemplificando con una actividad.

Cada práctica está representada en un formato, el cual consta de 5 partes que se describen a continuación:

<table>
<tr><td colspan="3">1) TEMA:</td></tr>
<tr><td>2) Práctica N° 1</td><td colspan="2">3) Instrucción</td></tr>
<tr><td colspan="2">4) Pseudocódigo o algoritmo</td><td>5) Programa</td></tr>
<tr><td colspan="2"></td><td></td></tr>
</table>

1) **Tema**: Tema que el alumno debe tener como antecedente. Tema al que hace referencia la práctica a programar.
2) **Actividad**: Número de práctica a programar.
3) **Instrucción**: Descripción de lo que se requiere que realice el alumno.
4) **Pseudocódigo o algoritmo**: Colocar el pseudocódigo (cuando las prácticas no sean muy amplias) o algoritmo (cuando las prácticas sean muy largas en líneas de instrucciones).
5) **Programa**: Pseudocódigo o algoritmo codificado en un lenguaje de programación estructurado.

Para el caso de que este problemario sea utilizado por los docentes, se sugiere que la parte 5 sea entregada, en caso de que ser muy amplia, en hojas anexas, y la parte 6 pueda ser entregada de manera escrita, en listado impreso o bien en un programa fuente como lo desee el docente que haga uso de este material.

El objetivo de este material es que los alumnos refuercen los conocimientos teóricos de cada tema de la unidad de aprendizaje, de esta manera los alumnos podrán aplicar las estructuras de datos dinámicas **pila, cola, lista, árbol y grafo** en la implementación de programas para la solución de problemas.

FORMA DE APLICACIÓN

Debido a que es una serie de ejercicios, su forma de aplicación no va más allá de entregarles el listado de prácticas a los alumnos para que ellos se dediquen a programarlas.

Puede ser aplicada en forma individual o bien en equipos, se recomienda en binas.

Dentro de la serie de prácticas se integran pictogramas las cuales se describen a continuación.

 Se refiere a que la práctica está resuelta en el apartado de prácticas resueltas.

 Su intención es informarle al alumno que a partir de que aparece este pictograma se introducen actividades con una nueva temática.

RECOMENDACIONES

Se recomienda a los alumnos:

- Que para evitar errores en la realización de prácticas, éste cuente con los conocimientos básicos referentes a cada tema en cuestión, es decir que se traten de realizar las prácticas una vez que previamente se estudiaron o abordaron los temas relacionados con éstas.
- Responderlas primeramente en pseudocódigo y posteriormente codificarlas.
- Si se llegase a tener alguna duda en la solución de las prácticas, el alumno debe recurrir al docente, a los apuntes o a la bibliografía sugerida.

EJEMPLOS DE SOLUCIÓN

TEMA: Aplicar la estructura de datos Pila Estática	
Ejemplo 1	**Programa que permita introducir 5 elementos de tipo entero en una pila estática y despliegue el elemento Tope**

Pseudocódigo o algoritmo

```
Pila: Registro
        Datos[5]: Entero
        Tope: Entero
Fin Registro
P : Pila
Programa
        I, Num: Entero
        Inicializa()
        I = 1
        Repite
                Escribir ("Dame número: ")
                Leer (Num)
                Push(Num)
                I=I+1
        Hasta (I>5)
        Escribir ("Elemento Tope ", Tope())
Termina
Modulo Inicializa ()
Inicio
        P.Tope = 0
Termina
Modulo Push (elem:Entero)
Inicio
        Si (PilaLlena() <> 1) entonces
                P.Tope = P.Tope+1
                P.datos[P.Tope] = elem
        Otro
                Escribir ("Pila llena")
        FinSi
Termina
Modulo PilaLlena (): Entero
Inicio
        Si (P.Tope = 5) entonces
                Regresa 1
        Otro
                Regresa 0
        FinSi
Termina
Modulo Tope(): Entero
Inicio
        Regresa P.datos[P.Tope]
Termina
```

Programa

```
#include <stdio.h>
struct Pila{
        int Datos[5];
        int Tope;
}
struct Pila P;
void Inicializa(), Push(int);
int PilaLlena(), Tope();
main(){
        int I, Num;
        Inicializa();
        I = 0;
        do{
                printf("Dame número: ");
                scanf ("%d", &Num);
                Push(Num);
                I=I+1;
        }while (I<5);
        printf ("Elemento Tope: %d ", Tope())
}
void Inicializa ()
{
        P.Tope = -1;
}
void Push (int elem)
{
        if (PilaLlena() ¡= 1) {
                P.Tope = P.Tope+1;
                P.datos[P.Tope] = elem;
        }
        else{
                printf ("Pila llena");
        }
}
int PilaLlena ()
{
        if (P.Tope == 4)
                return 1;
        else
                return 0;
}
int Tope()
{
        return P.datos[P.Tope];
}
```

TEMA: Aplicar la estructura de datos Cola Estática	
Ejemplo 2	**Almacenar los nombres de 5 profesores, desplegar el primer profesor**

Pseudocódigo o algoritmo

```
Cola: Registro
        Datos[5][30]: Cadena
        Fr: Entero
        Fi: Entero
Fin Registro
C : Cola
Programa
        Nom[30]: Cadena
        Op: Entero
        Inicializa()
        Repite
                Escribir (1. Insertar 2. Desplegar 3.
Salir")
                Leer(op)
                Caso (op)
                Op = 1:Escribir ("Dame nombre")
                        Leer (Nom)
                        Insertar(Nom)
                Op = 2:  Escribir ("Primer
                        Profesor", C.datos[C.Fr])
                FinCaso
        Hasta (Op = 3)
Termina
Modulo Inicializa ()
Inicio
        C.Fr = 0
        C.Fi = 0
Termina
Modulo Insertar (N[30]:Cadena)
Inicio
        Si (ColaLlena() <> 1) entonces
                C.Fi = C.Fi+1
                C.datos[C.Fi] = N
                Si (C.Fr = 0) entonces
                        C.Fr = 1
                FinSI
        Otro
                Escribir ("Cola llena")
        FinSi
Termina
Modulo ColaLlena (): Entero
Inicio
        Si (C.Fi = 5) entonces
                Regresa 1
        Otro
                Regresa 0
        FinSi
Termina
```

Programa

```
struct Cola{
        char Datos[5][30];
        int Fr;
        int Fi;
};
struct Cola C;
void inicializa(), Insertar(char);
int ColaLlena();
main(){
        char Nom[30];
        int Op;
        Inicializa();
        do{
                puts("1. Insertar 2. Desplegar 3.
Salir")
                scanf("%d", &op);
                switch (op){
                case 1: puts("Dame nombre");
                        gets(Nom);
                        Insertar(Nom);
                case 2:  puts ("Primer
                        Profesor %s",
                        C.datos[C.Fr]);
                }
        }while (Op ¡= 3)
}
void Inicializa ()
{
        C.Fr = -1;
        C.Fi = -1;
}
void Insertar (char N[30])
{
        if (ColaLlena() ¡= 1) {
                C.Fi = C.Fi+1;
                strcpy(C.datos[C.Fi],N);
                if (C.Fr == 0) C.Fr = 1;
        }
        else{
                puts ("Cola llena");
        }
}
int ColaLlena ()
{
        if (C.Fi == 4) return 1;
        else return 0;
}
```

TEMA: Aplicar la estructura de datos LDE	
Ejemplo 3	**Almacenar los datos de los medicamentos de una farmacia (Clave, descripción, precio) sin orden al final de la LDE. Realizar consultas de inicio a fin y fin a inicio a través de un menú: 1. Insertar 2. Desplegar Inicio-Fin 3. Desplegar Fin-Inicio 4. Salir**

Pseudocódigo o algoritmo

```
Nodo: Registro
        Cve: Entero
        Des[30]: Cadena
        Precio: Real
        *ant, *sig:Nodo
FinRegistro
*Inicio, *Final, *Nuevo: Nodo
Principal
Inicio
        Repite
                Escribir("1. Insertar 2.Desplegar Inicio-Fin 3.
Desplegar Fin-Inicio 4. Salir")
                Leer(op)
                Caso(op)
                        op=1: Insertar()
                        op=2: DesplegarIF()
                        op=3: DesplegarFI()
                FinCaso
        Hasta (op=4)
Termina
Modulo Inicializar()
Inicio
        Inicio = Final = Nulo
FinModulo
Modulo Insertar()
Inicio
        Nuevo=asignar memoria
        Escribir("Clave: ") Leer(Nuevo->Cve)
        Escribir("Descripción: ") Leer(Nuevo->Des)
        Escribir("Precio: ") Leer(Nuevo->Precio)
        Nuevo->sig = Nulo
        Nuevo->ant = Final
        Final = Nuevo
        Si (Inicio = Nulo) entonces
                Inicio = Nuevo
        FinSi
FinModulo
Modulo DesplegarIF()
Inicio
        *aux:Nodo
        aux=Inicio
        Mientras(aux <> Nulo)
                Escribir("Clave: ", aux->Cve)
                Escribir("Descripción: ",aux ->Des)
                Escribir("Precio: ",aux ->Precio)
                aux = aux->sig
        FinMientras
FinModulo
Modulo DesplegarFI()
Inicio
        *aux:Nodo
        aux=Final
        Mientras(aux <> Nulo)
                Escribir("Clave: ", aux->Cve)
                Escribir("Descripción: ",aux ->Des)
                Escribir("Precio: ",aux ->Precio)
                aux = aux->ant
        FinMientras
FinModulo
```

Programa

```
struct Nodo{
        int Cve;
        char Des[30];
        float Precio;
        struct Nodo *ant, *sig;
};
struct Nodo *Inicio, *Final, *Nuevo;
void Inicializar(), DesplegarIF(), DesplegarFI();
main(){
        do{
                printf("1. Insertar 2.Desplegar Inicio-Fin 3.
Desplegar Fin-Inicio 4. Salir");
                scanf("%d", &op);
                switch(op){
                        case 1: Insertar();
                        case 2: DesplegarIF();
                        case 3: DesplegarFI();
                }
        }while(op¡=4);
}
void Inicializar(){
        Inicio = Final = NULL;
}
void Insertar(){
        Nuevo=(struct Nodo *) malloc(sizeof(struct Nodo));
        printf ("Clave: ") scanf ("%d", Nuevo->Cve);
        printf ("Descripción: ") scanf ("%s", Nuevo->Des) ;
        printf ("Precio: ") scanf ("%f", Nuevo->Precio) ;
        Nuevo->sig = Nulo;
        Nuevo->ant = Final;
        Final = Nuevo;
        if (Inicio = Nulo) Inicio = Nuevo;
}
void DesplegarIF(){
        struct Nodo *aux;
        aux=Inicio;
        while (aux ¡= NULL){
                printf ("Clave: ", aux->Cve);
                printf ("Descripción: ",aux ->Des) ;
                printf ("Precio: ",aux ->Precio) ;
                aux = aux->sig;
        }
}
void DesplegarFI(){
        struct Nodo *aux;
        aux=Final;
        while (aux ¡= NULL){
                printf ("Clave: ", aux->Cve);
                printf ("Descripción: ",aux ->Des) ;
                printf ("Precio: ",aux ->Precio) ;
                aux = aux->ant;
        }
}
```

TEMA: Aplicar la estructura de datos Árbol	
Ejemplo 4	**Almacenar los datos de los productos de una ferretería (Clave, descripción, precio) en una estructura de árbol. Realizar consultas por orden de clave con base en un menú: 1. Insertar 2. Desplegar 3. Salir**

Pseudocódigo o algoritmo

```
Nodo: Registro
        Cve: Entero
        Des[30]: Cadena
        Precio: Real
        *I, *D:Nodo
FinRegistro
*Raíz, *Nuevo: Nodo
Principal
Inicio
        Repite
                Escribir("1. Insertar 2. Desplegar 3. Salir")
                Leer(op)
                Caso(op)
                        op=1: Insertar()
                        op=2: Desplegar(Raíz)
                FinCaso
        Hasta (op=3)
Termina
Modulo Inicializar()
Inicio
        Raíz = Nulo
FinModulo
Modulo Insertar()
Inicio
        Nuevo=asignar memoria
        Escribir("Clave: ") Leer(Cve)
        Escribir("Descripción: ") Leer(Des)
        Escribir("Precio: ") Leer(Precio)
        InsertarNodo(Nuevo, Cve, Des, Precio)
FinModulo
Módulo InsertarNodo(*N: Nodo,Cve:Entero,Des[30]:Cadena,Precio:Real)
Inicia
        Si (N <> Nulo) entonces
                Si (Cve < N->Cve) entonces
                        InsertarNodo(N->I, Cve,Des,Precio)
                Otro
                        Si (Cve > N->Num) entonces
                                Insertar (N->D,
Cve,Des,Precio)
                        Otro
                                Escribir ("Repetido")
                        FinSi
                FinSi
        Otro
                Nuevo=asignar memoria
                Nuevo->I = Nulo
                Nuevo->D = Nulo
                Nuevo->Cve = Cve
                Nuevo->Des = Des
                Nuevo->Precio = Precio
                N = Nuevo
        FinSi
FinMódulo

Modulo Desplegar(*N:Nodo)
Inicio
        Si (N <> Nulo) entonces
                Escribir (N ->Cve)
                Escribir (N ->Des)
                Escribir (N ->Precio)
                Desplegar(N->I)
                Desplegar(N->D)
        FinSi
FinModulo
```

Programa

```
struct Nodo{
        int Cve;
        char Des[30];
        float Precio;
        struct Nodo *I, *D;
}
struct Nodo *Raíz, *Nuevo;
void Inicializar(), Desplegar(struct Nodo *), Insertar(),
InsertarNodo(struct Nodo*, int, char, float);
main(){
        do{
                puts("1. Insertar 2. Desplegar 3. Salir");
                scanf("%d",op);
                switch(op){
                        case 1: Insertar()
                        case 2: Desplegar(Raíz)
                }
        }while(op!=3);
}
void Inicializar(){
        Raíz = Nulo;
}
void Insertar(){
        Nuevo=(struct Nodo *) malloc(sizeof(struct Nodo));
        puts ("Clave: "); scanf ("%d", Cve);
        puts ("Descripción: "); scanf ("%s", Des);
        puts ("Precio: "); scanf ("%f", Precio);
        InsertarNodo(Nuevo, Cve, Des, Precio);
}
void InsertarNodo (struct Nodo *N,int Cve, char Des[30], float,Precio){
        if (N ¡= NULL){
                if (Cve < N->Cve)
                        InsertarNodo(N->I,
Cve,Des,Precio);
                else{
                        if (Cve > N->Num)
                                Insertar (N->D,
Cve,Des,Precio);
                        else
                                puts ("Repetido");
                }
        }
        else{
                Nuevo=(struct Nodo *) malloc(sizeof(struct
Nodo));
                Nuevo->I = NULL;
                Nuevo->D = NULL;
                Nuevo->Cve = Cve;
                strcpy(Nuevo->Des = Des);
                Nuevo->Precio = Precio;
                N = Nuevo;
        }
}

void Desplegar (struct Nodo *N){
        if (N ¡= NULL){
                printf ("Clave: %d", N->Cve);
                printf ("Descripción: %s", N->Des);
                printf ("Precio: %f", N->Precio);
                Desplegar(N->I);
                Desplegar(N->D);
        }
}
```

TEMA: Aplicar la estructura de datos Grafo Estático	
Ejemplo 5	**Con base en un menú: 1. Insertar 2. Desplegar 3. Salir, utilizando un grafo estático, insertar los costos de boleto de avión entre tres destinos y desplegar su matriz de adyacencia**

Pseudocódigo o algoritmo

```
Grafo: Registro
    V[3]: Entero
    MA[3][3]: Real
FinRegistro
G:Grafo
Principal
Inicio
    Repite
        Escribir ("1. Insertar 2. Desplegar 3. Salir")
        Leer (op)
        Caso(op)
            op=1: Insertar()
            op=2: Desplegar()
        FinCaso
    Hasta(op=3)
Termina
Modulo Insertar()
Inicio
    InsertarVertice()
    InsertarArista()
FinModulo
Modulo InsertarVertice()
Inicio
    Para (i=1, i<4, i=i+1)
        Escribir ("Dame el nombre del vértice: ", i)
        Leer (G.V[i])
    FinPara
FinModulo
Modulo InsertarArista()
Inicio
    Para (i=1, i<4, i=i+1)
        Para (j=1, j<4, j=j+1)
            Escribir ("Dame el costo del boleto entre el vértice: ", i, " y el
vértice: ", j)
            Leer (G.MA[i][j])
        FinPara
    FinPara
FinModulo
Modulo Desplegar()
    Para (i=1, i<4, i=i+1)
        Para (j=1, j<4, j=j+1)
            Escribir (G.MA[i][j])
        FinPara
        Escribir (Salto de línea)
    FinPara
FinModulo
```

Programa

```
struct Grafo{
    int V[3];
    int MA[3][3];
};
struct Grafo G;
void insertar(), InsertarVertices(), InsertarAristas(), Desplegar();
main(){
    do{
        puts ("1. Insertar 2. Desplegar 3. Salir");
        scanf("%d", &op);
        switch(op){
            case 1: Insertar();
            case 2: Desplegar();
        }
    }while(op!=3);
}
void Insertar(){
    InsertarVertice();
    InsertarArista();
}
void InsertarVertice(){
    int i;
    for (i=0, i<3, i++){
        printf ("Dame el nombre del vértice: %d", i);
        scanf("%d", (G.V[i]);
    }
}
void InsertarArista(){
    int i, j;
    for (i=0; i<3; i++){
        for (j=0; j<3; j++){
            printf("Dame el costo del boleto entre el vértice: %d y
del vértice %d", i, j);
            scanf ("%f", &G.MA[i][j]);
        }
    }
}
void Desplegar(){
    int i,j;
    for (i=0; i<3; i++){
        for (j=0; j<3; j++){
            printf ("%f ", G.MA[i][j]);
        }
        printf ("\n");
    }
}
```

SERIE DE PRÁCTICAS

APLICAR LA ESTRUCTURA DE DATOS PILA

TEMA: Aplicar la estructura de datos Pila Estática	
Práctica 1	**Programa que permita introducir 5 elementos de tipo entero en una pila estática y despliegue sus elementos al irlos eliminando**
Pseudocódigo o algoritmo	**Programa**

Ejercicio resuelto página **30**

TEMA: Aplicar la estructura de datos Pila Estática	
Práctica 2	**Introducir 10 elementos de tipo entero en una pila estática y mediante la implementación de Pop ir sacando los elementos e introducirlos en una segunda pila. Desplegar ambas pilas**
Pseudocódigo o algoritmo	**Programa**

TEMA: Aplicar la estructura de datos Pila Estática	
Práctica 3	**Pedir 5 nombres de países, insertarlos en una pila, copiar esa pila en una segunda con los elementos en mayúsculas, desplegar las dos pilas**
Pseudocódigo o algoritmo	**Programa**

TEMA: Aplicar la estructura de datos Pila Estática	
Práctica 4	**Mediante un menú:** **1. Insertar 2. Eliminar 3. Desplegar 4. Salir** **Implementar las operaciones de una pila de máximo 10 elementos de tipo cadena que permita almacenar los colores introducidos por el usuario**
Pseudocódigo o algoritmo	**Programa**

TEMA: Aplicar la estructura de datos Pila Estática	
Práctica 5	**Llenar una pila de 15 elementos de tipo entero que sean generados aleatoriamente, desplegar la pila**
Pseudocódigo o algoritmo	**Programa**

TEMA: Aplicar la estructura de datos Pila Estática	
Práctica 6	**Mediante un menú:** **a. Insertar b. Eliminar b. Tope d. Salir** **Implementar las operaciones de una pila de máximo 10 elementos. La operación Insertar, inserta elementos de tipo entero generados aleatoriamente. La operación Tope despliega el elemento que se encuentra en el Tope de la pila.**
Pseudocódigo o algoritmo	**Programa**

TEMA: Aplicar la estructura de datos Pila Estática	
Práctica 7	**Se requiere guardar información de 5 alumnos cada uno con su cuenta, nombre y una calificación. Mediante la estructura de datos de pila estática guardar esta información y presentarla a manera de tabla**
Pseudocódigo o algoritmo	**Programa**

TEMA: Aplicar la estructura de datos Pila Estática	
Práctica 8	**Llenar dos pilas de 10 elementos aleatorios de tipo entero, generar una tercera pila con la combinación de ambas, introduciendo primero las de la pila 1 y después los de la pila 2, utilizando la operación push y pop para el manejo de las tres pilas, desplegar la pila 3**
Pseudocódigo o algoritmo	**Programa**

TEMA: Aplicar la estructura de datos Pila Estática	
Práctica 9	**Llenar dos pilas de 5 elementos aleatorios de tipo entero, generar una tercera pila con la combinación de ambas, introduciendo un elemento de la pila 1 y después un elemento de la pila 2, así sucesivamente, utilizando la operación push y pop para el manejo de las tres pilas, desplegar la pila 3**
Pseudocódigo o algoritmo	**Programa**

APLICAR LA ESTRUCTURA DE DATOS COLA

TEMA: Aplicar la estructura de datos Cola Estática	
Práctica 10	**En un banco se requiere formar los clientes que van llegando para ser atendidos con una ficha. Mediante las operaciones de una cola simular la llegada y atención de los clientes, con base en un menú: a. Formar b. Atender c. Salir**

Pseudocódigo o algoritmo	Programa

TEMA: Aplicar la estructura de datos Cola Estática	
Práctica 11	**En una tortillería se forman los clientes (20 máximo, utilizar nombre del cliente) que van llegando para ser atendidos por dos personas las cuales van atendiendo uno y uno. Mediante las operaciones de una cola simular la llegada y atención de los clientes, almacenando el número de cliente que fue atendido por cada persona, con base en un menú: a. Formar b. Atender c. Desplegar clientes d. Salir. La función Atender, permite asignar a cada persona un cliente, Desplegar clientes, debe desplegar los clientes de cada persona (2 colas)**

Pseudocódigo o algoritmo	Programa

TEMA: Aplicar la estructura de datos Cola Estática	
Práctica 12	**Con una cola estática de 5 elementos de tipo entero generados aleatoriamente implementar las operaciones con base en el menú: a. Insertar b. Eliminar c. Desplegar d. Salir**

Pseudocódigo o algoritmo	Programa

TEMA: Aplicar la estructura de datos Cola Estática	
Práctica 13	**Con una cola estática de 5 elementos de tipo entero generados aleatoriamente implementar las operaciones con base en el menú: a. Insertar b. Eliminar c. Desplegar d. Salir La función Desplegar, desplegará los elementos en forma horizontal diciendo cuál es el elemento de frente y cuál es el elemento final**
Pseudocódigo o algoritmo	**Programa**

TEMA: Aplicar la estructura de datos Cola Estática	
Práctica 14	**Con base en el menú: a. Insertar b. Eliminar c. Desplegar d. Estadísticas e. Salir, implementar las operaciones de una cola estática de 10 elementos de tipo entero introducidos por el usuario. La función Estadísticas debe desplegar el número de elementos insertados, número de espacios vacíos para ser ocupados y número de espacios desperdiciados**
Pseudocódigo o algoritmo	**Programa**

TEMA: Aplicar la estructura de datos Cola Estática	
Práctica 15	**Los 15 alumnos de una materia están formados para ser atendidos para la revisión de su calificación por parte de su profesor quien a su vez va formando a sus alumnos en dos filas con base a su calificación obtenida (aprobados y reprobados). Utilizando las operaciones de una cola estática desplegar las dos filas de reprobados y aprobados. En la inserción de los datos de cada alumno pedir su nombre y calificación**
Pseudocódigo o algoritmo	**Programa**

TEMA: Aplicar la estructura de datos Cola Estática	
Práctica 16	**Los 60 alumnos de un kinder van a ser distribuidos en 4 grupos, el grupo 1 va de la letra A-D, el segundo de la E-L, el tercero de la M-P y el cuarto de la Q-Z. Con una cola estática simular la formación y la distribución de los alumnos, imprimir los alumnos que tiene cada grupo**
Pseudocódigo o algoritmo	**Programa**

TEMA: Aplicar la estructura de datos Cola Estática	
Práctica 17	**En una farmacia llegaron 3 cajas de medicamentos, el empleado de dicha farmacia va acomodar los medicamentos en tres anaqueles alfabéticamente por laboratorio. Con una cola estática simular la distribución de los anaqueles (anaquel 1: A-F, anaquel 2: G-P y anaquel 3: Q-Z) e imprimir los medicamentos de cada uno de estos. La información que se tiene del medicamento es clave, nombre y laboratorio**
Pseudocódigo o algoritmo	**Programa**

TEMA: Aplicar la estructura de datos Cola Estática	
Práctica 18	**Con una cola estática de 10 elementos de tipo entero generados aleatoriamente implementar las operaciones con base en el menú: a. Insertar b. Eliminar c. Desplegar Frente y Final d. Salir**
Pseudocódigo o algoritmo	**Programa**

Ejercicio resuelto página **31**

APLICAR LA ESTRUCTURA DE DATOS LISTA

TEMA: Aplicar la estructura de datos LSE	
Práctica 19	**Se requiere almacenar los datos de los perros de una veterinaria (Número de identificación, raza, peso) con base en su número de identificación. Mediante una LSE insertar ordenadamente de menor a mayor los datos de cada animal y desplegarlos**
Pseudocódigo o algoritmo	**Programa**

Ejercicio resuelto página **32**

TEMA: Aplicar la estructura de datos LSE	
Práctica 20	**Mediante una LSE almacenar el número de cuenta, nombre y calificación de cada alumno de las materias de Cálculo I y Programación. Cada materia está representada por una LSE. Desplegar los datos de los alumnos que cursan las dos materias**
Pseudocódigo o algoritmo	**Programa**

TEMA: Aplicar la estructura de datos LDE	
Práctica 21	**Una LDE es utilizada para almacenar los datos de los profesores como son RFC, nombre, materia y teléfono. A través de un menú: 1. Insertar 2. Eliminar 3. Desplegar 4. Salir administrar los datos de los profesores de una Universidad**
Pseudocódigo o algoritmo	**Programa**

TEMA: Aplicar la estructura de datos LDE	
Práctica 22	**Mediante una LDE se almacenan ordenadamente los datos de los productos que se vende en una tienda comercial (clave, descripción y precio). A través de una menú: 1. Insertar 2. Eliminar 3. Buscar 4. Desplegar 5. Salir se administran estos productos**
Pseudocódigo o algoritmo	**Programa**

TEMA: Aplicar la estructura de datos LCSE	
Práctica 23	**Se quiere simular un radio mediante las estaciones de radio almacenadas ordenadamente con una LCSE. Mediante un menú: a. Insertar estación b. Eliminar Estación c. Radio (submenú: 1. Siguiente 2. Anterior 3. Terminar) d. Salir. En cada opción desplegar la estación que se está tocando en ese momento.**
Pseudocódigo o algoritmo	**Programa**

TEMA: Aplicar la estructura de datos LCDE	
Práctica 24	**Una LCDE se utiliza para simular el juego de una ruleta. Mediante un menú: a. Insertar números b. Eliminar números c. Jugar d. Salir implementar esa simulación. La función Juego pide el número a jugar y el número de vueltas que la ruleta dará a partir del último elemento de la LCD hacia atrás, esta función dirá si el usuario ganó o perdió**
Pseudocódigo o algoritmo	**Programa**

TEMA: Aplicar la estructura de datos LCDE	
Práctica 25	**Mediante las operaciones de una LCDE se quieren insertar los nombres de colores dados por un usuario. Implementar las funciones del siguiente menú: 1. Insertar 2. Desplegar de inicio a fin 3. Desplegar de fin a inicio 4. Salir**
Pseudocódigo o algoritmo	**Programa**

 Ejercicio resuelto página **34**

TEMA: Aplicar la estructura de datos Pila Dinámica	
Práctica 26	**Con la implementación de una pila dinámica, llenar dos pilas de 5 elementos aleatorios de tipo entero, generar una tercera pila con la combinación de ambas, introduciendo un elemento de la pila 1 y después un elemento de la pila 2, así sucesivamente, utilizando la operación push y pop para el manejo de las tres pilas, desplegar la pila 3**
Pseudocódigo o algoritmo	**Programa**

TEMA: Aplicar la estructura de datos Cola Dinámica	
Práctica 27	**Utilizando una cola dinámica, implementar el siguiente programa: los 15 alumnos de una materia están formados para ser atendidos para la revisión de su calificación por parte de su profesor quien a su vez va formando a sus alumnos en dos filas con base a su calificación obtenida (aprobados y reprobados). Desplegar las dos filas de reprobados y aprobados. En la inserción de los datos de cada alumno pedir su nombre y calificación**
Pseudocódigo o algoritmo	**Programa**

APLICAR LA ESTRUCTURA DE DATOS ÁRBOL

TEMA: Aplicar la estructura de datos Árbol	
Práctica 28	**Almacenar los promedios finales de un grupo de alumnos utilizando un árbol y desplegar las calificaciones con un recorrido en orden**
Pseudocódigo o algoritmo	**Programa**

Ejercicio resuelto página **35**

TEMA: Aplicar la estructura de datos Árbol	
Práctica 29	**Mediante un árbol binario de búsqueda, almacenar N números aleatorios y desplegar el árbol con recorrido en anchura. N es dado por el usuario**
Pseudocódigo o algoritmo	**Programa**

TEMA: Aplicar la estructura de datos Árbol	
Práctica 30	**Las votaciones de diferentes partidos son almacenados en un árbol binario de búsqueda, decir cuál fue el número de votaciones mayor y cuál fue el menor**
Pseudocódigo o algoritmo	**Programa**

TEMA: Aplicar la estructura de datos Árbol	
Práctica 31	**Los nombres de los profesores de una escuela son almacenados alfabéticamente en una estructura de árbol, desplegar los nombres alfabéticamente**
Pseudocódigo o algoritmo	**Programa**

TEMA: Aplicar la estructura de datos Árbol	
Práctica 32	**Desplegar la raíz del árbol binario de búsqueda en el que se almacenaron N números aleatorios y son controlados mediante el siguiente menú: 1. Insertar 2. Eliminar 3. Desplegar Raíz 4. Salir**
Pseudocódigo o algoritmo	**Programa**

TEMA: Aplicar la estructura de datos Árbol	
Práctica 33	**Desplegar todos los nodos terminales de un árbol binario de búsqueda en el que se almacenaron números introducidos por el usuario y son controlados mediante el siguiente menú: 1. Insertar 2. Eliminar 3. Desplegar Nodos terminales 4. Salir**
Pseudocódigo o algoritmo	**Programa**

TEMA: Aplicar la estructura de datos Árbol	
Práctica 34	**Leer una expresión aritmética y desplegar los recorridos de orden, preorden y suborden utilizando un árbol binario**
Pseudocódigo o algoritmo	**Programa**

TEMA: Aplicar la estructura de datos Árbol	
Práctica 35	**El RFC, nombre y salario de cada uno de los empleados de una fábrica son almacenados en un árbol binario de búsqueda con base en el salario, decir cuál es el empleado que más gana**
Pseudocódigo o algoritmo	**Programa**

TEMA: Aplicar la estructura de datos Árbol	
Práctica 36	**Se almacenan los datos de los empleados de una fábrica (RFC, nombre, turno y salario. Decir cuántos trabajan en el turno matutino, cuántos en el turno vespertino, utilizando la estructura de árbol binario de búsqueda**
Pseudocódigo o algoritmo	**Programa**

APLICAR LA ESTRUCTURA DE DATOS GRAFO

TEMA: Aplicar la estructura de datos Grafo Estático	
Práctica 37	**Se quieren almacenar las rutas de camiones de 3 destinos y el costo del boleto de cada ruta. Desplegar los datos almacenados mediante una matriz de adyacencia**
Pseudocódigo o algoritmo	**Programa**

Ejercicio resuelto página **36**

TEMA: Aplicar la estructura de datos Grafo Estático	
Práctica 38	**Leer los elementos de una matriz de 4x4 que representan los litros de agua que recorren de un pozo a otro (1-4), desplegar el conjunto de vértices y aristas del grafo formado por la matriz**
Pseudocódigo o algoritmo	**Programa**

TEMA: Aplicar la estructura de datos Grafo Estático	
Práctica 39	**Almacenar las distancias que se tienen entre 5 comunidades del estado. Con un menú: 1. Comunidades 2. Distancias 3. Verificar camino 4. Salir. Implementar las operaciones de un grafo estático. La función Verificar camino consiste en pedir las dos comunidades y decir si hay o no camino, si hay mostrar la distancia**
Pseudocódigo o algoritmo	**Programa**

TEMA: Aplicar la estructura de datos Grafo Estático	
Práctica 40	**Implementar las operaciones de un grafo estático para administrar las rutas de avión de una aerolínea, considerando 4 destinos y el tiempo de vuelo entre destinos. Desplegar la matriz de adyacencia**
Pseudocódigo o algoritmo	**Programa**

TEMA: Aplicar la estructura de datos Grafo Dinámico	
Práctica 41	**Con un grafo dinámico, almacenar las rutas de camiones de tres destinos y el costo del boleto de cada ruta. Implementar un menú: 1. Insertar Destino 2. Eliminar Destino 3. Insertar Costo 4. Eliminar Costo 5. Desplegar destinos y costos 6. Salir**
Pseudocódigo o algoritmo	**Programa**

TEMA: Aplicar la estructura de datos Grafo Dinámico	
Práctica 42	**Se quieren almacenar los litros de agua que recorren de un pozo a otro, mediante un grafo dinámico, decir cuál pozo tiene más conexiones a los demás**
Pseudocódigo o algoritmo	**Programa**

TEMA: Aplicar la estructura de datos Grafo Dinámico	
Práctica 43	**Con un grafo dinámico, almacenar las distancias que se tienen entre 5 comunidades del estado. Con un menú: 1. Comunidades 2. Distancias 3. Verificar camino 4. Salir. La función Verificar camino consiste en pedir las dos comunidades y decir si hay o no camino, si hay mostrar la distancia**
Pseudocódigo o algoritmo	**Programa**

TEMA: Aplicar la estructura de datos Grafo Dinámico	
Práctica 44	**Utilizando las operaciones de un grafo dinámico, administrar las rutas de avión de una aerolínea, considerando 4 destinos y el tiempo de vuelo entre destinos. Desplegar la matriz de adyacencia**
Pseudocódigo o algoritmo	**Programa**

TEMA: Aplicar la estructura de datos Grafo Dinámico	
Práctica 45	**El tráfico de datos entre los servidores en una red está representado por un grafo dinámico, decir cuál es el servidor que más tráfico recibe**
Pseudocódigo o algoritmo	**Programa**

PRÁCTICAS RESUELTAS

TEMA: Aplicar la estructura de datos Pila Estática

Práctica 1	**Programa que permita introducir 5 elementos de tipo entero en una pila estática y despliegue sus elementos al irlos eliminando**

Pseudocódigo o algoritmo

```
Pila: Registro
        Datos[5]: Entero
        Tope: Entero
Fin Registro
P : Pila
Programa
        I, Num: Entero
        Inicializa()
        I = 1
        Repite
                Escribir ("Dame número: ")
                Leer (Num)
                Push(Num)
                I=I+1
        Hasta (I>5)
        Desplegar()
Termina
Modulo Inicializa ()
Inicio
        P.Tope = 0
Termina
Modulo Push (elem:Entero)
Inicio
        Si (PilaLlena() <> 1) entonces
                P.Tope = P.Tope+1
                P.datos[P.Tope] = elem
        Otro
                Escribir ("Pila llena")
        FinSi
Termina
Modulo PilaLlena (): Entero
Inicio
        Si (P.Tope = 5) entonces
                Regresa 1
        Otro
                Regresa 0
        FinSi
Termina
Modulo Desplegar()
Inicio
    Para i=1, i<=P.Tope, i=i+1
        Escribir(Tope())
        Pop()
    FinPara
Termina
Modulo Tope()
Inicio
        Regresa (P.datos[P.Tope])
FinModulo
Modulo Pop()
Inicio
        Si (PilaVacia()<>1) entonces
                P.Tope=P.Tope-1
        Otro
                Escribir("Pila Vacía")
        FinSi
FinModulo
Modulo Pilavacia()
Inicio
        Si(P.Tope=-1) entonces regresa 1
        Otro regresa 0
FinModulo
```

Programa

```
#include <stdio.h>
struct Pila{
        int Datos[5];
        int Tope;
};
struct Pila P;
void Inicializa(), Push(int), Pop(), Desplegar();
int PilaLlena(), PilaVacia(), Tope();
main(){
        int I, Num;
        Inicializa();
        I = 0;
        do{
                printf("Dame número: ");
                scanf ("%d", &Num);
                Push(Num);
                I=I+1;
        }while (I<5);
        Despelgar();
}
void Inicializa (){
        P.Tope = -1;
}
void Push (int elem)
{
        if (PilaLlena() ¡= 1) {
                P.Tope = P.Tope+1;
                P.datos[P.Tope] = elem;
        }
        else{
                printf ("Pila llena");
        }
}
int PilaLlena (){
        if (P.Tope == 4) return 1;
        else return 0;
}
void Desplegar(){
    int i;
    for (i=1, i<=P.Tope; i++){
        printf("%d",Tope());
        Pop();
    }
}
int Tope(){
        return P.datos[P.Tope];
}
void Pop(){
        if (PilaVacia() ¡= 1) P.Tope--;
        else puts("Pila Vacía");
}
int PilaVacia(){
        if (P.Tope = = -1) return 1;
        else retiurn 0;
}
```

TEMA: Aplicar la estructura de datos Cola Estática	
Práctica 18	**Con una cola estática de 10 elementos de tipo entero generados aleatoriamente implementar las operaciones con base en el menú: a. Insertar b. Eliminar c. Desplegar Frente y Final d. Salir**

Pseudocódigo o algoritmo

```
Cola: Registro
        Datos[5][30]: Cadena
        Fr: Entero
        Fi: Entero
Fin Registro
C : Cola
Programa
        Num, Op: Entero
        Inicializa()
        Repite
                Escribir (a. Insertar b. Eliminar c. Desplegar d.
Salir")
                Leer(Op)
                Caso (Op)
                        Op = 1: Num = aleatorio
                        Insertar(Num)
                        Op = 2: Eliminar()
                        Op = 3: Desplegar()
                FinCaso
        Hasta (Op = 4)
Termina
Modulo Inicializa ()
Inicio
        C.Fr = 0
        C.Fi = 0
Termina
Modulo Insertar (Num: Entero)
Inicio
        Si (ColaLlena() <> 1) entonces
                C.Fi = C.Fi+1
                C.Datos[C.Fi] = Num
                Si (C.Fr = 0) entonces
                        C.Fr = 1
                FinSi
        Otro
                Escribir ("Cola llena")
        FinSi
Termina
Modulo ColaLlena (): Entero
Inicio
        Si (C.Fi = 10) entonces
                Regresa 1
        Otro
                Regresa 0
        FinSi
Termina
Modulo Eliminar()
Inicio
        Si (ColaVacia() <> 1) entonces
                C.Fr = C.Fr+1
        Otro
                Escribir ("Cola Vacía")
        FinSi
Termina
Modulo ColaVacia(): Entero
Inicio
        Si (C.Fi=0 y C.Fr=0) o (C.Fi < C.Fr) entonces
                regresa 1
        otro
                regresa 0
        FinSi
Termina
Modulo Desplegar()
Inicio
        Escribir("Frente: ", C.Datos[C.Fr])
        Escribir("Final: ", C.Datos[C.Fi])
Termina
```

Programa

```
struct Cola{
        char Datos[5][30];
        int Fr;
        int Fi;
};
struct Cola C;
void inicializa(), Insertar(int), Eliminar(),Desplegar();
int ColaLlena(), ColaVacia();
main(){
        int Op, Num;
        Inicializa();
        do{
                puts("a. Insertar b. Eliminar c. Desplegar d.
Salir")
                scanf("%d", &op);
                switch (op){
                case 'a':       Num = srandom();
                                Insertar(Num);
                case 'b':       Eliminar();
                case 'c':       Desplegar();
                }
        }while (Op ¡= 3);
}
void Inicializa (){
        C.Fr = -1;
        C.Fi = -1;
}
void Insertar (int Num){
        if (ColaLlena() ¡= 1) {
                C.Fi = C.Fi+1;
                C.Datos[C.Fi] = Num;
                if (C.Fr == -1) C.Fr = 0;
        }
        else{
                puts ("Cola llena");
        }
}
int ColaLlena ()
{
        if (C.Fi == 9) return 1;
        else return 0;
}
void Eliminar(){
        if (ColaVacia() ¡= 1) C.Fr = C.Fr+1;
        else printf ("Cola Vacía");
}
int  ColaVacia(){
        if (C.Fi=-1 y C.Fr=-1) || (C.Fi < C.Fr) return 1;
        else return 0;
}
void Desplegar(){
        printf ("Frente: %d", C.Datos[C.Fr]);
        printf ("Final: %d", C.Datos[C.Fi]);
}
```

TEMA: Aplicar la estructura de datos LSE	
Práctica 19	**Se requiere almacenar los datos de los perros de una veterinaria (Número de identificación, raza, peso) con base en su número de identificación. Mediante una LSE insertar ordenadamente de menor a mayor los datos de cada animal y desplegarlos**

Pseudocódigo o algoritmo

```
Nodo:Registro
   Id: Entero
   Raza[10]: Cadena
   Peso: Real
   *sig: Nodo
FinRegistro
*Inicio, *Nuevo: Nodo
Principal
Inicio
   Inicializar()
   Repite
      Escribir (“1. Insertar 2. Desplegar 3. Salir”)
      Leer (op)
      Caso (op)
         Op = 1: Escribir (“Identificacion: “) Leer(Id)
                 Escribir (“Raza: “) Leer(Raza)
                 Escribir (Peso: “) Leer (Peso)
                 Insertar(Id, Raza, Peso)
         Op = 2: Desplegar()
      FinCaso
   Hasta (op = 3)
Termina
Modulo Inicializar()
Inicio
   Inicio = Nulo
FinModulo
Modulo Insertar(Id:Entero, Raza[10]:Cadena, Peso:Real)
Inicio
   Nuevo = Asignar Memoria
   Nuevo->Id = Id
   Nuevo->Raza = Raza
   Nuevo->Peso = Peso
   Si (Inicio = Nulo) entonces
      Nuevo->sig = Nulo
      Inicio = Nuevo
   Otro
      Si (Inicio->Id > Id) entonces
         InsertarInicio()
      Otro
         Ult = BuscarUltimo()
         Si (Ult->Id < Id) entonces
            InsertarFinal(Ult)
         Otro
            InsertarEntreNodos(Id)
         FinSi
      FinSi
   FinSi
FinModulo
Modulo InsertarInicio()
   Nuevo->sig = Inicio
   Inicio = Nuevo
FinModulo
Modulo InsertarFinal(*Ult:Nodo)
   Ult->sig = Nuevo
   Nuevo->sig = Nulo
FinModulo
Modulo InsertarEntreNodos(Id: Entero)
   Pos = BuscarPosicion(Id)
   Nuevo->sig = Pos->sig
   Pos->sig = Nuevo
FinModulo
Modulo *BuscarUltimo(): Nodo
Inicio
   Aux=Inicio
   Mientras(Aux->sig <> Nulo)
      Aux = Aux->sig
   FinMientras
   Regresa Aux
```

Programa

```
struct Nodo{
   int Id;
   char Raza[10];
   float Peso;
   struct Nodo *sig;
FinRegistro
struct Nodo *Inicio, *Nuevo;
main(){
   Inicializar();
   do{
      puts (“1. Insertar 2. Desplegar 3. Salir”);
      scanf (“%d”, &op);
      switch (op){
         case 1: printf (“Identificacion: “);
                 scanf(“%d”, &Id);
                 printf (“Raza: “);
                 scanf(“%s”, Raza);
                 printf (Peso: “);
                 scanf(“%f”,&Peso);
                 Insertar(Id, Raza, Peso);
         case 2: Desplegar();
      }
   }while (op ¡= 3);
}
void Inicializar(){
   Inicio = NULL;
}
void Insertar(Id:Entero, Raza[10]:Cadena, Peso:Real){
   Nuevo = (struct Nodo *) malloc(sizeof(struct Nodo));
   Nuevo->Id = Id;
   Nuevo->Raza = Raza;
   Nuevo->Peso = Peso;
   if (Inicio = Nulo){
      Nuevo->sig = NULL;
      Inicio = Nuevo;
   }
   else{
      if (Inicio->Id > Id)
         InsertarInicio();
      else{
         Ult = BuscarUltimo();
         if (Ult->Id < Id)
            InsertarFinal(Ult);
         else
            InsertarEntreNodos(Id) ;
      }
   }
}
void InsertarInicio(){
   Nuevo->sig = Inicio;
   Inicio = Nuevo;
}
void InsertarFinal(struct Nodo *Ult){
   Ult->sig = Nuevo;
   Nuevo->sig = Nulo;
}
void InsertarEntreNodos(int Id){
   Pos = BuscarPosicion(Id) ;
   Nuevo->sig = Pos->sig;
   Pos->sig = Nuevo;
}
struct Nodo *BuscarUltimo(){
   Aux=Inicio;
   while (Aux->sig ¡= Nulo){
      Aux = Aux->sig;
   }
   return Aux;
}
```

```
FinModulo
Modulo *BuscarPosicion(Id: Entero)
Inicio
   Aux = Inicio
   Aux2 = Aux
   Mientras (Aux->Id > Id y Aux <> Nulo)
      Aux2 = Aux
      Aux = Aux->sig
   FunMientras
   Si (Aux = Nulo) entonces
      regresa Aux
   Otro
      Regresa Aux2
   FinSi
FinModulo
Modulo Desplegar()
Inicio
   Aux = Inicio
   Mientras (Aux <> Nulo)
      Escribir ("Numero de Identificación: ", Aux ->Id)
      Escribir ("Raza: ", Aux ->Raza)
      Escribir ("Peso: ", Aux ->Peso)
      Aux = Aux->sig
   FinMientras
FinModulo
```

```
struct Nodo *BuscarPosicion(int Id){
   Aux = Inicio;
   Aux2 = Aux;
   while (Aux->Id > Id && Aux ¡= NULL){
      Aux2 = Aux;
      Aux = Aux->sig;
   }
   if (Aux = Nulo) return Aux;
   else reurn Aux2;
}
void Desplegar(){
   Aux = Inicio;
   while (Aux <> Nulo){
      printf ("Numero de Identificación: %d", Aux ->Id) ;
      printf ("Raza: %s", Aux ->Raza) ;
      printf ("Peso: %f", Aux ->Peso) ;
      Aux = Aux->sig;
   }
}
```

TEMA: Aplicar la estructura de datos LCDE	
Práctica 25	**Mediante las operaciones de una LCDE se quieren insertar los nombres de colores dados por un usuario. Implementar las funciones del siguiente menú: 1. Insertar 2. Desplegar de inicio a fin 3. Desplegar de fin a inicio 4. Salir**

Pseudocódigo o algoritmo

```
Nodo:Registro
   Color[15]: Cadena
   *ant, *sig: Nodo
FinRegistro
*Lc, *Nuevo: Nodo
Principal
Inicio
   Inicializar()
   Repite
      Escribir (“1. Insertar 2. Desplegar Inicio-Fin 3.
DesplegarFinal-Inicio 4. Salir”)
      Leer (op)
      Caso (op)
         Op = 1: Escribir (“Color: “)
               Leer (Color)
               InsertarFinal(Color)
         Op = 2: DesplegarIF()
         Op = 3: DesplegarFI()
      FinCaso
   Hasta (op = 4)
Termina
Modulo Inicializar()
Inicio
   Lc = Nulo
FinModulo
Modulo InsertarFinal(Color[15]:Cadena)
Inicio
   Nuevo = Asignar Memoria
   Nuevo->Color = Color
   Si (Lc = Nulo) entonces
      Nuevo->ant = Nulo
      Nuevo->sig = Nulo
   Otro
      Nuevo->ant = Lc
      Nuevo->sig = Lc->sig
   FinSi
   Lc = Nuevo
FinModulo
Modulo DesplegarIF()
Inicio
   Aux = Lc->sig
   Mientras (Aux <> Lc)
      Escribir (Aux ->Color)
      Aux = Aux->sig
   FinMientras
FinModulo
Modulo DesplegarFI()
Inicio
   Aux = Lc
   Mientras (Aux <> Lc->sig)
      Escribir (Aux ->Color)
      Aux = Aux->ant
   FinMientras
   Escribir (Aux ->Color)
FinModulo
```

Programa

```
struct Nodo{
   char Color[15];
   struct Nodo *ant, *sig;
}
struct Nodo *Lc, *Nuevo;
main(){
   Inicializar();
   do{
      puts (“1. Insertar 2. Desplegar Inicio-Fin 3.
DesplegarFinal-Inicio 4. Salir”);
      scanf (“%d”, &op);
      switch (op){
         case 1: puts (“Color: “);
               gets (Color);
               InsertarFinal(Color);
         case 2: DesplegarIF();
         case 3: DesplegarFI();
      }
   }while (op < 4)
}
void Inicializar(){
   Lc = NULL;
}
void InsertarFinal(char Color[15]){
   Nuevo = (struct Nodo *) malloc(sizeof(struct Nodo));
   strcpy(Nuevo->Color, Color);
   if (Lc == NULL) {
      Nuevo->ant = Nulo;
      Nuevo->sig = Nulo;
   }
   else{
      Nuevo->ant = Lc;
      Nuevo->sig = Lc->sig;
   }
   Lc = Nuevo;
}
void DesplegarIF(){
   Aux = Lc->sig;
   while(Aux <> Lc){
      printf (“%s\n“, Aux ->Color);
      Aux = Aux->sig;
   }
}
void DesplegarFI(){
   Aux = Lc;
   while(Aux <> Lc->sig){
      printf(“%s\n“, Aux ->Color);
      Aux = Aux->ant;
   }
   printf(“%s\n“, Aux ->Color);
}
```

TEMA: Aplicar la estructura de datos Árbol	
Práctica 28	**Almacenar los promedios finales de un grupo de alumnos utilizando un árbol y desplegar las calificaciones con un recorrido en orden**

Pseudocódigo o algoritmo

```
Nodo: Registro
        Cal: Real
        *I, *D:Nodo
FinRegistro
*Raíz, *Nuevo: Nodo
Principal
Inicio
        Repite
                Escribir("1. Insertar 2. Desplegar 3. Salir")
                Leer(op)
                Caso(op)
                        op=1: Insertar()
                        op=2: Desplegar(Raíz)
                FinCaso
        Hasta (op=3)
Termina
Modulo Inicializar()
Inicio
        Raíz = Nulo
FinModulo
Modulo Insertar()
Inicio
        Nuevo=asignar memoria
        Escribir("Calificación: ")
        Leer(Calif)
        InsertarNodo(Nuevo, Calif)
FinModulo
Módulo InsertarNodo(*N: Nodo, Calif:Real)
Inicia
        Si (N <> Nulo) entonces
                Si (Calif < N->Cal) entonces
                        InsertarNodo(N->I, Calif)
                Otro
                        Si (Calif > N->Cal) entonces
                                Insertar (N->D, Calif)
                        Otro
                                Escribir ("Repetido")
                        FinSi
                FinSi
        Otro
                Nuevo=asignar memoria
                Nuevo->I = Nulo
                Nuevo->D = Nulo
                Nuevo->Cal = Calif
                N = Nuevo
        FinSi
FinMódulo

Modulo Desplegar(*N:Nodo)
Inicio
        Si (N <> Nulo) entonces
                Desplegar(N->I)
                Escribir (N->Cal)
                Desplegar(N->D)
        FinSi
FinModulo
```

Programa

```
struct Nodo{
        float Cal;
        struct Nodo *I, *D;
}
struct Nodo *Raíz, *Nuevo;
void Inicializar(), Desplegar(struct Nodo *), Insertar(),
InsertarNodo(struct Nodo*, float);
main(){
        do{
                puts("1. Insertar 2. Desplegar 3. Salir");
                scanf("%d",op);
                switch(op){
                        case 1: Insertar()
                        case 2: Desplegar(Raíz)
                }
        }while(op<3);
}
void Inicializar(){
        Raíz = Nulo;
}
void Insertar(){
        Nuevo=(struct Nodo *) malloc(sizeof(struct Nodo));
        puts ("Calificación: ");
        scanf ("%f", Calif);
        InsertarNodo(Nuevo, Calif);
}
void InsertarNodo (struct Nodo *N, float Calif){
        if (N ¡= NULL){
                if (Calif < N->Cal)
                        InsertarNodo(N->I, Cal);
                else{
                        if (Calif > N->Cal)
                                Insertar (N->D,
Calif);
                        else
                                puts ("Repetido");
                }
        }
        else{
                Nuevo=(struct Nodo *)
malloc(sizeof(struct Nodo));
                Nuevo->I = NULL;
                Nuevo->D = NULL;
                Nuevo->Cal = Calif;
                N = Nuevo;
        }
}

void Desplegar (struct Nodo *N){
        if (N ¡= NULL){
                Desplegar(N->I);
                printf (" %f-", N->Cal);
                Desplegar(N->D);
        }
}
```

TEMA: Aplicar la estructura de datos Grafo Estático	
Práctica 37	**Se quieren almacenar las rutas de camiones de 3 destinos y el costo del boleto de cada ruta. Desplegar los datos almacenados mediante una matriz de adyacencia**

Pseudocódigo o algoritmo

```
Grafo: Registro
    V[3]: Entero
    MA[3][3]: Real
FinRegistro
G:Grafo
Principal
Inicio
    Repite
        Escribir (“1. Insertar 2. Desplegar 3. Salir”)
        Leer (op)
        Caso(op)
            op=1: Insertar()
            op=2: Desplegar()
        FinCaso
    Hasta(op=3)
Termina
Modulo Insertar()
Inicio
    InsertarVertice()
    InsertarArista()
FinModulo
Modulo InsertarVertice()
Inicio
    Para (i=1, i<4, i=i+1)
        Escribir (“Dame el nombre del destino: “, i)
        Leer (G.V[i])
    FinPara
FinModulo
Modulo InsertarArista()
Inicio
    Para (i=1, i<4, i=i+1)
        Para (j=1, j<4, j=j+1)
            Escribir (“Dame el costo del boleto entre el destino: “, i, “ y
el destino: “, j)
            Leer (G.MA[i][j])
        FinPara
    FinPara
FinModulo
Modulo Desplegar()
    Para (i=1, i<4, i=i+1)
        Para (j=1, j<4, j=j+1)
            Escribir (G.MA[i][j])
        FinPara
        Escribir (Salto de línea)
    FinPara
FinModulo
```

Programa

```
struct Grafo{
    int V[3];
    int MA[3][3];
};
struct Grafo G;
void insertar(), InsertarVertices(), InsertarAristas(), Desplegar();
main(){
    do{
        puts (“1. Insertar 2. Desplegar 3. Salir”);
        scanf(“%d”, &op);
        switch(op){
            case 1: Insertar();
            case 2: Desplegar();
        }
    }while(op!=3);
}
void Insertar(){
    InsertarVertice();
    InsertarArista();
}
void InsertarVertice(){
    int i;
    for (i=0, i<3, i++){
        printf (“Dame el nombre del destino: %d“, i);
        scanf(“%d”, (G.V[i]);
    }
}
void InsertarArista(){
    int i, j;
    for (i=0; i<3; i++){
        for (j=0; j<3; j++){
            printf(“Dame el costo del boleto entre el destino: %d y
del destino %d“, i+1, j+1);
            scanf (“%f”, &G.MA[i][j]);
        }
    }
}
void Desplegar(){
    int i,j;
    for (i=0; i<3; i++){
        for (j=0; j<3; j++){
            printf (“%f ”, G.MA[i][j]);
        }
        printf (“\n”);
    }
}
```

BIBLIOGRAFÍA

BÁSICA

Araujo Serna Lourdes, Martínez Unanue Raquel, Rodríguez Artacho Miguel. (2011). Programación y estructuras de datos avanzadas. Editorial Universitaria Ramon Areces.

Cairó, Osvaldo y Guardati, Silvia. (2006). Estructuras de datos (3a. Edición). McGraw-Hill.

Criado, Ma. Asunción. (2006). Programación en lenguajes estructurados. AlfaOmega Ra-Ma.

Franch Gutiérrez, Xavier. Estructuras de datos. Especificación, diseño e implementación, Ediciones de la UPC, S.L., 2004.

Garrido, Antonio y Fernández Joaquín, Abstracción y Estructuras de Datos en C++, Delta Publicaciones, 2006.

Joyanes, Aguilar Luis; Zahonero, Martínez Ignacio. (2004). Estructura de Datos. Algoritmos y estructuras de datos: una perspectiva en C. McGraw-Hill, Madrid.

Joyanes, Luis. (2008). Fundamentos de programación (4ª Edición). McGraw-Hill.

Joyanes Aguilar Luis, Zahonero Ignacio. (2005). Programación en C: metodología, algoritmos y estructuras de datos. McGraw-Hill

Libardo Pantoja César Pardo. (2017). Estructuras de datos dinámicas: una forma fácil de aprender. Ra-Ma S.A. Editorial y Publicaciones.

López, Leobardo. (2004). Programación estructurada. Un enfoque algorítmico (2ª. Edición). AlfaOmega.

Luján Mora Sergio, Ferrández Rodríguez Antonio, Peral Cortés Jesús, Requena Jiménez Antonio. (2014). Ejercicios resueltos sobre Programación y estructuras de datos. Universidad de Alicante.

Narciso Martí Oliet, Yolanda Ortega Mallén, José Alberto Verdejo López. (2004). Estructuras de datos y métodos algorítmicos: ejercicios resueltos. Pearson Educación

Rodríguez Artalejo, González Caldero, Gómez Martin. (2011). Estructuras de datos. Un enfoque moderno, Editorial Complutense.

Rudolph Russell. (2018). Estructuras de Datos y Algoritmos: Una Introducción Sencilla. CreateSpace Independent Publishing Platform

COMPLEMENTARIA

Drozdeck, Adam. (2007). Estructuras de datos y algoritmos en Java (2ª Edición). Thomson.

Garrido Carrillo Antonio, Fernández-Valdivia Joaquín. (2006). Abstracción y estructuras de datos en C++. Delta Publicaciones.

Goodrich Michael T., Tamassia Roberto, Goldwasser Michael H. (2014). Data Structures and Algorithms in Java. John Wiley & Sons.

Joyanes, Luis. M. Fernández, L: Sánchez, I. Zahonero. (2005). Estructuras de datos en C. McGraw-Hill. Schaum.

Koffman, Elliot y Wolfgang, Paul. (2008). Estructura de datos con C++. Objetos, abstracciones y diseño. McGraw-Hill.

Nyhoff, Larry. (2006). TADs, Estructuras de datos y resolución de problemas con C++. Pearson - Prentice Hall.

ANEXOS

PILA

Operación	Algoritmo	Especificaciones	Implementación
Insertar (Push)	1. Verificar si la pila no está llena 2. Incrementar en 1 el puntero (tope) de la pila 3. Almacenar el elemento nuevo en la posición del puntero de la pila	Verificar que la pila no esté llena antes de intentar insertar un elemento. Si está llena el programa debe enviar un mensaje de error y el programa debe terminar su ejecución	***Modulo Insertar (dato: E)*** *Inicio* *Si (p->tope = MAX) entonces* *Escribir ("Pila llena")* *Otro* *p->tope ← p->tope + 1* *p->datos[p->tope] ← dato* *FinSi* *Termina* O bien ***Modulo Insertar(*p: pila, dato: E)*** *Inicio* *Si (PilaLlena(p) = 1) entonces* *Escribir ("Pila llena")* *Otro* *p->tope ← p->tope + 1* *p->datos[p->tope] ← dato* *FinSi* *Termina*
Quitar (Pop)	1. Verificar si la pila no está vacía 2. Leer el elemento de la posición del puntero (tope) de la pila 3. Decrementar en 1 el tope de la pila	Verificar que la pila no esté vacía antes de intentar quitar un elemento. Si está vacía el programa debe enviar un mensaje de error y el programa debe terminar su ejecución	***Modulo Quitar( )*** *Inicio* *Si (PilaVacia(p) = 1) entonces* *Escribir ("Pila Vacía")* *Otro* *p->tope ← p->tope - 1* *FinSi* *Termina*
Pila Vacía	1. Verificar si el puntero de la pila vale 0	Devuelve 1 (verdadero) si la pila está vacía y 0 (falso) en caso contrario.	***Modulo PilaVacia(*p: pila): E*** *Inicio* *Si (p->tope = 0) entonces* *regresa 1*

Operación	Algoritmo	Especificaciones	Implementación
			Otro *regresa 0* *FinSi* *Termina*
Pila Llena	1. Verificar si el puntero de la pila vale el número máximo de elementos permitidos en el arreglo especificado en la declaración de la pila.	Devuelve 1 (verdadero) si la pila está llena y 0 (falso) en caso contrario.	***Modulo PilaLlena(*p: pila): E*** *Inicio* *Si (p->tope = MAX) entonces* *regresa 1* *Otro* *regresa 0* *FinSi* *Termina*
Inicializar	1. Asignar al puntero de la pila (tope) el valor de 0.	Se limpia o vacía la pila, dejándola sin elementos.	***Modulo Inicializar(*p : pila)*** *Inicio* *p->tope ← 0* *Termina*
Cima	1. Verificar si la pila no está vacía. 2. Leer el elemento situado en la posición especificado por el puntero de la pila (tope) en el arreglo.	Si la pila no está vacía, devuelve el valor situado en la cima de la pila, pero no decrementa el puntero de la pila ya que la pila queda intacta. Si está vacía regresa el valor de 0.	***Modulo Cima(*p: pila)*** *Inicio* *Si (PilaVacia(p) = 1) entonces* *regresa 0* *Otro* *regresa p->datos[p->tope]* *FinSi* *Termina*

COLA

Operación	Algoritmo	Especificaciones	Implementación
Insertar	1. Verificar si la cola no está llena 2. Almacenar el elemento nuevo en la posición del puntero final de la cola 3. Incrementar en 1 el puntero del final de la cola	Verificar que la cola no esté llena antes de intentar insertar un elemento. Si está llena el programa debe enviar un mensaje de error y el programa debe terminar su ejecución	***Modulo Insertar (dato:E)*** *Inicio* *Si(ColaLlena(c)=1) entonces* *Escribir("Cola llena")* *Otro* *c->datos[c->final] ← dato* *c->final ← c->final + 1* *FinSi* *Termina*
Quitar	1. Verificar si la cola no está vacía 2. Leer el elemento de la posición del puntero (frente) de la cola 3. Incrementa en 1 el frente de la cola	Verificar que la cola no esté vacía antes de intentar quitar un elemento. Si está vacía el programa debe enviar un mensaje de error y el programa debe terminar su ejecución	***Modulo Quitar( )*** *Inicio* *Si (ColaVacia(c) = 1) entonces* *Escribir ("Cola Vacía")* *Otro* *c->frente ← c->frente + 1* *FinSi* *Termina*
Cola Vacía	1. Verificar si el frente de la cola es igual al final de la cola	Devuelve 1 (verdadero) si la cola está vacía y 0 (falso) en caso contrario.	***Modulo ColaVacia(*c: cola) : E*** *Inicio* *Si ((c->final=1 y c->frente=1) o (c->final < c->frente)) entonces* *Regresa 1* *Otro* *Regresa 0* *FinSi* *Termina*

Operación	Algoritmo	Especificaciones	Implementación
Cola Llena	1. Verificar si el final de la cola es mayor que el número máximo de elementos permitidos en el arreglo especificado en la declaración de la cola.	Devuelve 1 (verdadero) si la cola está llena y 0 (falso) en caso contrario.	***Modulo ColaLlena(*c: cola): E*** *Inicio* *Si (c->final > MAX) entonces* *regresa 1* *Otro* *regresa 0* *FinSi* *Termina*
Inicializar	1. Asignar tanto al puntero frente como final de la cola el valor de 1.	Se limpia o vacía la cola, dejándola sin elementos.	***Modulo Inicializar(*c :cola)*** *Inicio* *c->frente ← 1* *c->final ← 1* *Termina*
Frente	1. Verificar si la cola no está vacía. 2. Leer el elemento situado en la posición especificado por el puntero frente de la cola en el arreglo.	Si la cola no está vacía, devuelve el valor situado en el frente de la cola, pero se no incrementa el puntero frente de la cola ya que la cola queda intacta. Si está vacía regresa el valor de 0.	***Modulo Frente(*c: cola)*** *Inicio* *Si (ColaVacia(c) = 1) entonces* *regresa 0* *Otro* *regresa c->datos[c->frente]* *FinSi* *Termina*

LISTA SIMPLEMENTE ENLAZADA

Insertar

```
Módulo Insertar (valor:E, *L: Nodo)
Inicio
        *N: Nodo
        N ← Asignación de memoria
        Si (N ≠ Nulo) entonces
            Si (L = Nulo) entonces
                N->x ← valor
                N->sig ← Nulo
                L ← N
                cabeza ← L
            Otro
                InsertarFin(valor,N)
            FinSi
        Otro
            Escribir("Error en Memoria")
        FinSi
Termina
```

InsertarFin

```
Módulo InsertarFin (valor:E, *N: Nodo)
Inicio
    *aux: Nodo
    aux ← Ultimo( )
    N->x ← valor
    N->sig ← Nulo
    aux-> sig ← N
Termina
```

InsertarPrim

```
Módulo InsertarPrim (valor:E, *N: Nodo)
Inicio
    N->x ← valor
    N->sig ← cabeza
    cabeza ← N
Termina
```

Eliminar

```
Módulo Eliminar (valor:E, *L:Nodo)
Inicio
    *aux: Nodo
    Si (L ≠ Nulo) entonces
        aux ← Buscar(valor)
        Si (aux ≠ Nulo) entonces
            Si (aux = cabeza) entonces
                EliminarPrim()
            Otro
                Si (aux->sig = Nulo) entonces
                    EliminarFin(aux)
                Otro
                    Eliminar2(aux)
                FinSi
            FinSi
        Otro
            Escribir ("Elemento no encontrado")
        FinSi
    Otro
        Escribir("Lista Vacía")
    FinSi
Termina
```

EliminarPrim	EliminarFin
Módulo EliminarPrim () ***Inicio*** **aux: Nodo* *aux ← cabeza* *cabeza ← cabeza->sig* *Liberar (aux)* ***Termina***	***Módulo EliminarFin (*N:Nodo)*** ***Inicio*** **aux: Nodo* *aux ← Anterior(N)* *aux->sig ← Nulo* *Liberar (N)* ***Termina***
EsVacía	**ListaVacía**
Módulo EsVacía () ***Inicio*** *Si (cabeza = Nulo) entonces* *Regresa 1* *Otro* *Regresa 0* *FinSi* ***Termina***	***Módulo ListaVacía ()*** ***Inicio*** *cabeza ← Nulo* ***Termina***
Ultimo(Recorrer)	**Buscar**
Módulo Ultimo( ): Nodo ***Inicio*** **aux: Nodo* *aux ← cabeza* *Mientras (aux->sig ≠ Nulo)* *aux ← aux->sig* *FinMientras* *Regresa aux* ***Termina***	***Módulo Buscar (valor:E): Nodo*** ***Inicio*** **aux: Nodo* *aux ← cabeza* *Mientras(aux ≠ nulo)* *Si (aux->x ≠ valor) entonces* *aux ← aux->sig* *Otro* *Regresa aux* *FinSi* *FinMientras* *Regresa nulo* ***Termina***
Anterior	**Siguiente**
Módulo Anterior(*N:Nodo): Nodo ***Inicio*** **aux: Nodo* *aux ← cabeza* *Mientras (aux->sig ≠ N)* *aux ← aux->sig* *FinMientras* *Regresa aux* ***Termina***	***Módulo Siguiente (*N:Nodo): Nodo*** ***Inicio*** *Regresa N->sig* ***Termina***

Primero	Visualiza
Módulo Primero( ): Nodo ***Inicio*** *Regresa cabeza* ***Termina***	***Módulo Visualiza( )*** ***Inicio*** **N: Nodo* *N ← cabeza* *Mientras (N ≠ Nulo)* *Escribir(N->x)* *N ← N->sig* *FinMientras* ***Termina***
Insertar2	**Eliminar2**
Módulo Insertar2(*N:Nodo, *NIzq:Nodo, *NDer:Nodo) ***Inicio*** *N->sig ← NDer* *NIzq->sig ←N* ***Termina***	***Módulo Eliminar2(*N:Nodo)*** ***Inicio*** **aux: Nodo* *aux ← Anterior(N)* *aux->sig ← N->sig* *Liberar(N)* ***Termina***

LISTA DOBLEMENTE ENLAZADA

Insertar

```
Módulo Insertar (valor:E, *L: Nodo)
Inicio
        *N: Nodo
        N ← Asignación de memoria
        Si (N ≠ Nulo) entonces
            Si (L = Nulo) entonces
                N->x ← valor
                N->sig ← Nulo
                N->ant ← Nulo
                L ← N
                cabeza ← L
                final ← L
            Otro
                InsertarFin(valor,N)
            FinSi
        Otro
            Escribir("Error en Memoria")
        FinSi
Termina
```

InsertarFin

```
Módulo InsertarFin (valor:E, *N: Nodo)
Inicio
    N->x ← valor
    N->sig ← Nulo
    N->ant ← final
    final-> sig ← N
    final ← N
Termina
```

InsertarPrim

```
Módulo InsertarPrim (valor:E, *N: Nodo)
Inicio
    N->x ← valor
    N->ant ← Nulo
    N->sig ← cabeza
    cabeza-> anterior ← N
    cabeza ← N
Termina
```

Eliminar

```
Módulo Eliminar (valor:E, *L:Nodo)
Inicio
    *aux: Nodo
    Si (L ≠ Nulo) entonces
        aux ← Buscar(valor)
        Si (aux ≠ Nulo) entonces
            Si (aux = cabeza) entonces
                EliminarPrim()
            Otro
                Si (aux = final) entonces
                    EliminarFin()
                Otro
                    Eliminar2(aux)
                FinSi
            FinSi
        Otro
            Escribir ("Elemento no encontrado")
        FinSi
```

	Otro *Escribir("Lista Vacía")* *FinSi* ***Termina***
EliminarPrim	**EliminarFin**
Módulo EliminarPrim () ***Inicio*** **aux: Nodo* *aux ← cabeza* *cabeza ← cabeza->sig* *cabeza->anterior ← Nulo* *Liberar (aux)* ***Termina***	***Módulo EliminarFin ()*** ***Inicio*** **aux: Nodo* *aux ← final* *final ← final->anterior* *final->siguiente ← Nulo* *Liberar (aux)* ***Termina***
EsVacía	**ListaVacía**
Módulo EsVacía () ***Inicio*** *Si (cabeza = Nulo) entonces* *Regresa 1* *Otro* *Regresa 0* *FinSi* ***Termina***	***Módulo ListaVacía ()*** ***Inicio*** *cabeza ← Nulo* *final ← Nulo* ***Termina***
Ultimo	**Primero**
Módulo Ultimo( ): Nodo ***Inicio*** *Regresa final* ***Termina***	***Módulo Primero( ): Nodo*** ***Inicio*** *Regresa cabeza* ***Termina***
Anterior	**Siguiente**
Módulo Anterior(*N:Nodo): Nodo ***Inicio*** *Regresa N->ant* ***Termina***	***Módulo Siguiente (*N:Nodo): Nodo*** ***Inicio*** *Regresa N->sig* ***Termina***
Buscar	**Visualiza**
Módulo Buscar (valor:E): Nodo ***Inicio*** **aux: Nodo* *aux ← cabeza* *Mientras(aux ≠ nulo)* *Si (aux->x ≠ valor) entonces* *aux ← aux->sig* *Otro* *Regresa aux* *FinSi*	***Módulo Visualiza()*** ***Inicio*** **N: Nodo* *N ← cabeza* *Mientras (N ≠ Nulo)* *Escribir(N->x)* *N ← N->sig* *FinMientras* ***Termina***

FinMientras *Regresa nulo* ***Termina***	
Insertar2	**Eliminar2**
Módulo Insertar2(*N:Nodo, *NIzq:Nodo, *NDer:Nodo) ***Inicio*** *N->sig ← NDer* *N->ant ← NIzq* *NIzq->sig ← N* *NDer->ant ← N* ***Termina***	***Módulo Eliminar2(*N:Nodo)*** ***Inicio*** *N->ant->sig ← N->sig* *N->sig->ant ← N->ant* *Liberar(N)* ***Termina***

LISTA CIRCULAR SIMPLEMENTE ENLAZADA

Insertar

```
Módulo Insertar (valor:E, *L: ListaC)
Inicio
        *N: ListaC
        N ← Asignación de memoria
        Si (N ≠ Nulo) entonces
            Si (L = Nulo) entonces
                N->x ← valor
                N->sig ← N
                L ← N
                Lc ← L
            Otro
                InsertarFin(valor,N)
            FinSi
        Otro
            Escribir("Error en Memoria")
        FinSi
Termina
```

InsertarFin

```
Módulo InsertarFin (valor:E, *N: ListaC)
Inicio
    N->x ← valor
    N->sig ← Lc->sig
    Lc-> sig ← N
    Lc ← N
Termina
```

InsertarPrim

```
Módulo InsertarPrim (valor:E, *N: ListaC)
Inicio
    N->x ← valor
    N->sig ← Lc->sig
    Lc-> sig ← N
Termina
```

Eliminar

```
Módulo Eliminar (valor:E, *L:ListaC)
Inicio
    *aux: ListaC
    Si (L ≠ Nulo) entonces
        aux ← Buscar(valor)
        Si (aux ≠ Nulo) entonces
            Si (aux = Lc->sig) entonces
                EliminarPrim()
            Otro
                Si (aux = Lc) entonces
                    EliminarFin()
                Otro
                    Eliminar2(aux)
                FinSi
            FinSi
        Otro
            Escribir ("Elemento no encontrado")
        FinSi
```

	Otro *Escribir("Lista Vacía")* *FinSi* ***Termina***
EliminarPrim	**EliminarFin**
Módulo EliminarPrim () ***Inicio*** **aux: ListaC* *aux ← Lc->sig* *Lc->sig ← Lc->sig->sig* *Liberar (aux)* ***Termina***	***Módulo EliminarFin ()*** ***Inicio*** **aux: ListaC* *aux ← Anterior(Lc)* *aux->sig ← Lc->sig* *Liberar (Lc)* *Lc ← aux* ***Termina***

LISTA CIRCULAR DOBLEMENTE ENLAZADA

Insertar

```
Módulo Insertar (valor:E, *L: ListaCD)
Inicio
        *N: ListaCD
        N ← Asignación de memoria
        Si (N ≠ Nulo) entonces
            Si (L = Nulo) entonces
                N->x ← valor
                N->sig ← N
                N->ant ← N
                L ← N
                Lc ← L
            Otro
                InsertarFin(valor,N)
            FinSi
        Otro
            Escribir("Error en Memoria")
        FinSi
Termina
```

InsertarFin

```
Módulo InsertarFin (valor:E, *N: ListaCD)
Inicio
    N->x ← valor
    N->ant ← Lc
    N->sig ← Lc->sig
    Lc-> sig ← N
    Lc ← N
Termina
```

InsertarPrim

```
Módulo InsertarPrim (valor:E, *N: ListaCD)
Inicio
    N->x ← valor
    N->ant ← Lc
    N->sig ← Lc->sig
    Lc-> sig ← N
Termina
```

Eliminar

```
Módulo Eliminar (valor:E, *L:ListaCD)
Inicio
    *aux: ListaCD
    Si (L ≠ Nulo) entonces
        aux ← Buscar(valor)
        Si (aux ≠ Nulo) entonces
            Si (aux = Lc->sig) entonces
                EliminarPrim()
            Otro
                Si (aux = Lc) entonces
                    EliminarFin()
                Otro
                    Eliminar2(aux)
                FinSi
            FinSi
        Otro
            Escribir ("Elemento no encontrado")
        FinSi
    Otro
        Escribir("Lista Vacía")
    FinSi
Termina
```

EliminarPrim	EliminarFin
Módulo EliminarPrim () ***Inicio*** **aux: ListaCD* *aux ← Lc->sig* *Lc->sig->ant ← Lc* *Lc->sig ← Lc->sig->sig* *Liberar (aux)* ***Termina***	***Módulo EliminarFin ()*** ***Inicio*** **aux: ListaCD* *aux ← Lc->ant* *Lc->ant->sig ← Lc->sig* *Lc->sig->ant ← Lc->ant* *Liberar (Lc)* *Lc ← aux* ***Termina***

ÁRBOL

Operación	Implementación
Búsqueda	***Módulo Buscar (*N: Nodo, dato:E)*** ***Inicia*** *Si (N ≠ Nulo) entonces* *Si (dato < N->Num) entonces* *Buscar (N->I, dato)* *Otro* *Si (dato > N->Num) entonces* *Buscar (N->D, dato)* *Otro* *Escribir ("Elemento Encontrado")* *FinSi* *FinSi* *Otro* *Escribir ("Elemento No Encontrado")* *FinSi* ***FinMódulo***
Inserción	***Módulo Insertar (*N: Nodo, dato:E)*** ***Inicia*** *Si (N ≠ Nulo) entonces* *Si (dato < N->Num) entonces* *Insertar (N->I, dato)* *Otro* *Si (dato > N->Num) entonces* *Insertar (N->D, dato)* *Otro* *Escribir ("El Nodo ya se encuentra")* *FinSi* *FinSi* *Otro* *Crea (Nuevo)* *Nuevo->I ← Nulo* *Nuevo->D ← Nulo* *Nuevo->Num ← dato*

Operación	Implementación
	N ← Nuevo *FinSi* ***FinMódulo***
Eliminación	***Módulo Borrar (*N: Nodo)*** ***Inicia*** *Si (N ≠ Nulo) entonces* *Si (dato < N->Num) entonces* *Borrar (N->I, dato)* *Otro* *Si (dato > N->Num) entonces* *Borrar (N->D, dato)* *Otro* *Otro ← N* *Si (Otro->D = Nulo) entonces* *N ← Otro->I* *Otro* *Si (Otro->I = Nulo) entonces* *N ← Otro->D* *Otro* *Aux ← N->I* *Band ← 0* *Mientras (Aux->D ≠ Nulo)* *Aux1 ← Aux* *Aux ← Aux->D* *Band ← 1* *FinMientras* *N->Num ← Aux->Num* *Otro ← Aux* *Si (Band = 1) entonces* *Aux1->D ← Aux->I* *Otro* *N->I ← Aux->I* *FinSi* *FinSi* *FinSi* *Libera memoria (Otro)* *FinSi* *FinSi* *Otro* *Escribir (“Elemento No Encontrado”)* *FinSi* ***FinMódulo***

GRAFO ESTÁTICO

Operación	Descripción	Implementación
InicializaGrafo(G)	El grafo no tiene vértices ni arcos	***Módulo IniciaGrafo(*G: Grafo)*** ***Inicia*** *I, J: E* *Para I←1, I<N, I←I+1* *G->V[I] ← 0* *Para J←1, J<=N, J←J+1* *G->A[I][J] ← 0* *FinPara* *FinPara* ***Termina***
Union(u,v)	Añade el arco (u, v) al grafo G. Añadir los dos vértices del arco	***Módulo AnadeArco(*G: Grafo, Arc: Arco)*** ***Inicia*** *AnadeVertice(G, Arc.u)* *AnadeVertice(G, Arc.v)* *G->A[Arc.u][Arc.v] ← 1* ***Termina***
BorrarArco(u,v)	Elimina del grafo G el arco	***Módulo ElimArco(*G: Grafo, Arc: Arco)*** ***Inicia*** *G->A[Arc.u][Arc.v] ← 0* ***Termina***
Adyacente(u,v)	Función que devuelve cero (0) si no forman un arco los vértices u, v	***Módulo PerteneceArco(*G: Grafo, Arc: Arco)*** ***Inicia*** *Sw: E* *Sw ← PerteneceVertice(G, Arc.u) y* *PertenceVertice(G, Arc.v)* *Si (Sw = 1) entonces* *Regresa G->A[Arc.u][Arc.v]* *Otro* *Regresa 0* *FinSi* ***Termina***

Operación	Descripción	Implementación
	Decide si un vértice está en el grafo	***Módulo PerteneceVertice(*G: Grafo, Arc: Arco)*** ***Inicia*** *Si (v >= 1 y v<=N) entonces* *Regresa G->V[v]* *Otro* *Regresa 0* *FinSi* ***Termina***
NuevoVértice(v)	Añade el vértice *v* al grafo	***Módulo AnadeVertice(*G: Grafo, v:E)*** ***Inicia*** *Si (v >= 1 y v<=N) entonces* *G->V[v] ← 1* *FinSi* ***Termina***
BorrarVértice(v)	Elimina del grafo G el vértice v	***Módulo ElimVertice(*G: Grafo, v:E)*** ***Inicia*** *I: E* *Si (v >= 0 y v<N) entonces* *G->V[v] ← 0* *Para I←1, I<=N, I←I+1* *G->A[I][v] ← 0* *G->A[v][I] ← 0* *FinPara* *FinSi* ***Termina***

GRAFOS DINÁMICOS

Operación	Implementación
AñadeConjuntoAdy	***Módulo AñadeConjuntoAdy (*Primero: ListaAdy, dato:ItemAdy)*** ***Inicia*** **Nuevo: ListaAdy* *Si (PerteneceConjuntoAdy(Primero.dato) ≠ 1) entonces* *Nuevo ← Asignar memoria* *Nuevo->el ← dato* *Nuevo->sig ← Primero* *Primero ← Nuevo* *FinSi* ***Termina***
PerteneceConjuntoAdy	***Módulo PerteneceConjuntoAdy(*Primero: ListaAdy, dato: ItemAdy)*** ***Inicia*** **ptr: ListaAdy* *Para ptr←Primero, ptr ≠ nulo, ptr ← ptr->sig* *Si (ptr->el.v = dato.v) entonces* *Regresa 1* *FinSi* *FinPara* *Regresa 0* ***Termina***
BorraConjuntoAdy	***Módulo BorraConjuntoAdy(*Primero: ListaAdy, dato: ItemAdy)*** ***Inicia*** **ptr, *ant: ListaAdy* *EncuentraPosAdy(Primero, dato, ant, ptr)* *Si (ptr ≠ nulo) entonces* *Si (ptr = primero) entonces* *Primero ← ptr->sig* *Otro* *ant->sig ←ptr->sig* *FinSi* *FinSi* *Liberar (ptr)* ***Termina***
BorraListaAdy	***Módulo BorraListaAdy(*Primero: ListaAdy)*** ***Inicia*** **l, *ll: ListaAdy* *l ← Primero* *Mientras (l ≠ nulo)*

Operación	Implementación
	ll ← l *l ← l->sig* *Liberar (ll)* *FinMientras* ***Termina***
EncuentraPosAdy	***Módulo EncuentraPosAdy(*Primero: ListaAdy, dato: ItemAdy, *Ant: ListaAdy, *Pos: ListaAdy)*** ***Inicia*** **ptr, *antt: ListaAdy* *enc: E* *enc ← 0* *ptr ← Primero* *antt ← nulo* *Mientras (enc ≠ 1 y ptr ≠ nulo)* *Si (ptr->el-v = dato.v) entonces* *enc ← 1* *Otro* *enc ← 0* *FinSi* *Si (enc ≠ 1) entonces* *antt ←ptr* *ptr ← ptr->sig* *FinSi* *FinMientras* *Ant ← antt* *Pos ← ptr* ***Termina***
Inicializa	***Módulo Inicializa (*Primero: ListaG)*** ***Inicia*** *Primero ← Nulo* ***Termina***
AñadeArco	***Módulo AñadeArco(*Primero: ListaG, arc: Arco)*** ***Inicia*** *Dato: ItemG* *Dato1:ItemAdy* **Ant, *Pos: ListaG* *Dato.v ← arc.u* *Dato.Ady ← nulo* *AñadeVertice(Primero, Dato)*

Operación	Implementación
	Dato.v ← arc.v *AñadeVertice(Primero, Dato)* *Dato.v ← arc.u* *EncuentraPosGrafo(Primero, Dato,Ant,Pos)* *Dato1.v ← arc.v* *Dato1.valor ← arc.valor* *AñadeConjuntoAdy(Pos->el.Ady, Dato1)* ***Termina***
BorraArco	***Módulo BorraArco(*Primero: ListaG, arc: Arco)*** ***Inicia*** *Dato: ItemG* *Dato1:ItemAdy* **Ant, *Pos: ListaG* *Dato.v ← arc.u* *Dato.Ady ← nulo* *EncuentraPosGrafo(Primero, Dato,Ant,Pos)* *Dato1.v ← arc.v* *Dato1.valor ← arc.valor* *BorraConjuntoAdy(Pos->el.Ady, Dato1)* ***Termina***
AñadeVértice	***Módulo AñadeVertice(*Primero: ListaG, dato: ItemAdy)*** ***Inicia*** **Nuevo: ListaG* *Si (PertenceVertice(Primero,dato) ≠ 1) entonces* *Nuevo ← Asignar memoria* *Nuevo->el ← dato* *Nuevo->sig ← Primero* *Primero ← Nuevo* *FinSi* ***Termina***
BorraVértice	***Módulo BorraVertice(*Primero: ListaG, dato:ItemG)*** ***Inicia*** **ptr, *ant: ListaG* *Dato1: ItemAdy* *EncuentraPosGrafo(Primero, dato, ant, ptr)* *Si (ptr ≠ nulo) entonces* *Si (ptr = primero) entonces* *Primero ← ptr->sig* *Otro* *ant->sig ←ptr->sig* *FinSi*

Operación	Implementación
	BorraListaAdy(ptr->el.Ady) *Liberar (ptr)* *ptr ← primero* *dato1.v ← dato.v* *Mientras (ptr ≠ nulo)* *BorraConjuntoAdy(ptr->el.Ady),dato1)* *ptr ← ptr->sig* *FinMientras* *FinSi* ***Termina***
PerteneceVertice	***Módulo PerteneceVertice(*Primero: ListaG, dato: ItemAdy)*** ***Inicia*** **ptr: ListaG* *Para ptr←Primero, ptr ≠ nulo, ptr ← ptr->sig* *Si (ptr->el.v = dato.v) entonces* *Regresa 1* *FinSi* *FinPara* *Regresa 0* ***Termina***
PerteneceArco	***Módulo PerteneceArco (*Primero: ListaG, dato: ItemAdy)*** ***Inicia*** *Dato: ItemG* *Dato1:ItemAdy* **Ant, *Pos: ListaG* **Ant1, *Pos1: ListaAdy* *Dato.v ← arc.u* *Dato.Ady ← nulo* *EncuentraPosGrafo(Primero, Dato,Ant,Pos)* *Dato1.v ← arc.v* *Dato1.valor ← arc.valor* *Si (Pos ≠ nulo) entonces* *EncuentraPosAdy(Pos->el.Ady, Dato1,Ant1,Pos1)* *Si (Pos1 ≠ Nulo) entonces* *Regresa 1* *Otro* *Regresa 0* *FinSi* *FinSi* *Regresa 0* ***Termina***