



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

<b>Presentación.....</b>	<b>6</b>
<b>Guía de uso.....</b>	<b>8</b>
<b>Forma de Aplicación.....</b>	<b>9</b>
<b>Recomendaciones.....</b>	<b>9</b>
<b>Ejemplos de solución.....</b>	<b>10</b>
<b>Serie de prácticas.....</b>	<b>15</b>
<b>Aplicar la estructura de datos Pila.....</b>	<b>15</b>
Práctica 1. Programa que permita introducir 5 elementos de tipo entero en una pila estática y despliegue sus elementos al irlos eliminando.....	15
Práctica 2. Introducir 10 elementos en una pila estática y mediante la implementación de Pop ir sacando los elementos e introducirlos en una segunda pila. Desplegar ambas pilas.....	15
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.....	15
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.....	16
Práctica 5. Llenar una pila de 15 elementos de tipo entero que sean generados aleatoriamente, desplegar la pila.....	16
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.....	16
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.....	17
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.....	17
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.....	17
<b>Aplicar la estructura de datos Cola</b>	<b>18</b>
Práctica 10. En un banco se requiere formar los clientes (10 máximo) que van llegando para ser atendidos con una ficha. Mediante las operaciones de una cola simular	18

	la llegada y atención de los clientes, con base en un menú: a. Formar b. Atender c. Salir.....	
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).....	18
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.....	18
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.....	19
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.....	19
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.....	19
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.....	20
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.....	20
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.....	20
<b>Aplicar la estructura de datos Lista</b>		<b>21</b>
Práctica 19.	Se requiere almacenar los datos de los animales 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.....	21
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.....	21
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	21

	3. Desplegar 4. Salir administrar los datos de los profesores de una Universidad.....	
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.....	22
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.....	22
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 LCDE hacia atrás, esta función dirá si el usuario ganó o perdió.....	22
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.....	23
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.....	23
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.....	23
<b>Aplicar la estructura de datos Árbol</b>		<b>24</b>
Práctica 28.	Almacenar los promedios finales de un grupo de alumnos utilizando un árbol binario y desplegar las calificaciones con un recorrido en orden.....	24
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.....	24
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...	24
Práctica 31.	Los nombres de los profesores de una escuela son almacenados alfabéticamente en una estructura de árbol, desplegar los nombres alfabéticamente.....	25
Práctica 32.	Desplegar la raíz del árbol binario de búsqueda en el que se almacenaron números aleatorios y son controlados mediante el siguiente menú: 1. Insertar 2. Eliminar 3. Desplegar Raíz 4. Salir.....	25
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 .....	25
Práctica 34.	Leer una expresión aritmética y desplegar los recorridos de orden, preorden y suborden utilizando un árbol binario.....	26
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.....	26

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.....	26
<b>Aplicar la estructura de datos Grafo</b>		<b>27</b>
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.....	27
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.....	27
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.....	27
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.....	28
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.....	28
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	28
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.....	29
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 .....	29
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.....	29
<b>Prácticas Resueltas.....</b>		<b>30</b>
<b>Bibliografía.....</b>		<b>36</b>
<b>Anexos.....</b>		<b>39</b>
Pila.....		39
Cola.....		41
Lista.....		43
Árbol.....		53
Grafo.....		55

## 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:

<b>1) TEMA:</b>	
<b>2) Práctica N° 1</b>	<b>3) Instrucción</b>
<b>4) Pseudocódigo o algoritmo</b>	<b>5) Programa</b>

- 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	Programa
<p>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&gt;5)              Escribir (“Elemento Tope “, Tope())          Termina          Modulo Inicializa ()          Inicio              P.Tope = 0          Termina          Modulo Push (elem:Entero)          Inicio              Si (PilaLlena() &lt;&gt; 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</p>	<pre>#include &lt;stdio.h&gt; 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”, &amp;Num);         Push(Num);         I=I+1;     }while (I&lt;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]; }</pre>

TEMA: Aplicar la estructura de datos Cola Estática	
Ejemplo 2	Almacenar los nombres de 5 profesores, desplegar el primer profesor
Pseudocódigo o algoritmo	Programa
<p>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() &lt;&gt; 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</p>	<pre> 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”, &amp;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; } </pre>

**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**

**Programa**

```

    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
    
```

```

    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)
Inicio
    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
    Otro
        FinSi
        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()
Inicio
    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]);
        }
    }
}
    
```

# 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 un 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. Siguiendo 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

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

<b>TEMA: Aplicar la estructura de datos Árbol</b>	
<b>Práctica 32</b>	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
<b>Pseudocódigo o algoritmo</b>	<b>Programa</b>

<b>TEMA: Aplicar la estructura de datos Árbol</b>	
<b>Práctica 33</b>	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
<b>Pseudocódigo o algoritmo</b>	<b>Programa</b>

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

<b>TEMA: Aplicar la estructura de datos Árbol</b>	
<b>Práctica 35</b>	<b>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</b>
<b>Pseudocódigo o algoritmo</b>	<b>Programa</b>

<b>TEMA: Aplicar la estructura de datos Árbol</b>	
<b>Práctica 36</b>	<b>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</b>
<b>Pseudocódigo o algoritmo</b>	<b>Programa</b>



## 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	Programa
<p>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&gt;5)  Desplegar()  Termina  Modulo Inicializa ()  Inicio      P.Tope = 0  Termina  Modulo Push (elem:Entero)  Inicio      Si (PilaLlena() &lt;&gt; 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&lt;=P.Tope, i=i+1          Escribir(Tope())          Pop()      FinPara  Termina  Modulo Tope()  Inicio      Regresa (P.datos[P.Tope])  FinModulo  Modulo Pop()  Inicio      Si (PilaVacía()&lt;&gt;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</p>	<pre>#include &lt;stdio.h&gt; struct Pila{     int Datos[5];     int Tope; }; struct Pila P; void Inicializa(), Push(int), Pop(), Desplegar(); int PilaLlena(), PilaVacía(), Tope(); main(){     int I, Num;     Inicializa();     I = 0;     do{         printf("Dame número: ");         scanf ("%d", &amp;Num);         Push(Num);         I=I+1;     }while (I&lt;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&lt;=P.Tope; i++){         printf("%d",Tope());         Pop();     } } int Tope(){     return P.datos[P.Tope]; } void Pop(){     if (PilaVacía() != 1) P.Tope--;     else puts("Pila Vacía"); } int PilaVacía(){     if (P.Tope == -1) return 1;     else return 0; } }</pre>

**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**

Cola: Registro  
 Datos[5][30]: Cadena  
 Fr: Entero  
 Fi: Entero

Fin Registro  
 C : Cola  
 Programa

Num, Op: Entero  
 Inicializa()  
 Repite

Salir”) Escribir (a. Insertar b. Eliminar c. Desplegar d.

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 (ColaVacía() <> 1) entonces  
 C.Fr = C.Fr+1

Otro

Escribir (“Cola Vacía”)

FinSi

Termina  
 Modulo ColaVacía(): 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

```

struct Cola{
    char Datos[5][30];
    int Fr;
    int Fi;
};
struct Cola C;
void inicializa(), Insertar(int), Eliminar(),Desplegar();
int ColaLlena(), ColaVacía();
main(){
    int Op, Num;
    Inicializa();
    do{
        puts(“a. Insertar b. Eliminar c. Desplegar d.
        Salir”)
        scanf(“%d”, &op);
        switch (op){
            case ‘a’:    Num = random();
                        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 (ColaVacía() != 1) C.Fr = C.Fr+1;
    else printf (“Cola Vacía”);
}
int ColaVacía(){
    if (C.Fi==0 y C.Fr==0) || (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
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;
}
  
```

<pre> FinModulo Modulo *BuscarPosicion(Id: Entero) Inicio   Aux = Inicio   Aux2 = Aux   Mientras (Aux-&gt;Id &gt; Id y Aux &lt;&gt; Nulo)     Aux2 = Aux     Aux = Aux-&gt;sig   FinMientras   Si (Aux = Nulo) entonces     regresa Aux   Otro     Regresa Aux2   FinSi FinModulo Modulo Desplegar() Inicio   Aux = Inicio   Mientras (Aux &lt;&gt; Nulo)     Escribir ("Numero de Identificación: ", Aux -&gt;Id)     Escribir ("Raza: ", Aux -&gt;Raza)     Escribir ("Peso: ", Aux -&gt;Peso)     Aux = Aux-&gt;sig   FinMientras FinModulo </pre>	<pre> struct Nodo *BuscarPosicion(int Id){   Aux = Inicio;   Aux2 = Aux;   while (Aux-&gt;Id &gt; Id &amp;&amp; Aux != NULL){     Aux2 = Aux;     Aux = Aux-&gt;sig;   }   if (Aux = Nulo) return Aux;   else return Aux2; }  void Desplegar(){   Aux = Inicio;   while (Aux &lt;&gt; Nulo){     printf ("Numero de Identificación: %d", Aux -&gt;Id) ;     printf ("Raza: %s", Aux -&gt;Raza) ;     printf ("Peso: %f", Aux -&gt;Peso) ;     Aux = Aux-&gt;sig;   } } </pre>
---	---

**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**

```

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

```

```

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
    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, Calif);
        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()
Inicio
  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)	<ol style="list-style-type: none"> <li>1. Verificar si la pila no está llena</li> <li>2. Incrementar en 1 el puntero (tope) de la pila</li> <li>3. Almacenar el elemento nuevo en la posición del puntero de la pila</li> </ol>	<p>Verificar que la pila no esté llena antes de intentar insertar un elemento.</p> <p>Si está llena el programa debe enviar un mensaje de error y el programa debe terminar su ejecución</p>	<p><b>Modulo Insertar (dato: E)</b></p> <p>Inicio</p> <p>    Si <math>(p \rightarrow \text{tope} = \text{MAX})</math> entonces</p> <p>        Escribir ("Pila llena")</p> <p>    Otro</p> <p>        <math>p \rightarrow \text{tope} \leftarrow p \rightarrow \text{tope} + 1</math></p> <p>        <math>p \rightarrow \text{datos}[p \rightarrow \text{tope}] \leftarrow \text{dato}</math></p> <p>    FinSi</p> <p>Termina</p> <p>O bien</p> <p><b>Modulo Insertar(*p: pila, dato: E)</b></p> <p>Inicio</p> <p>    Si <math>(\text{PilaLlena}(p) = 1)</math> entonces</p> <p>        Escribir ("Pila llena")</p> <p>    Otro</p> <p>        <math>p \rightarrow \text{tope} \leftarrow p \rightarrow \text{tope} + 1</math></p> <p>        <math>p \rightarrow \text{datos}[p \rightarrow \text{tope}] \leftarrow \text{dato}</math></p> <p>    FinSi</p> <p>Termina</p>
Quitar (Pop)	<ol style="list-style-type: none"> <li>1. Verificar si la pila no está vacía</li> <li>2. Leer el elemento de la posición del puntero (tope) de la pila</li> <li>3. Decrementar en 1 el tope de la pila</li> </ol>	<p>Verificar que la pila no esté vacía antes de intentar quitar un elemento.</p> <p>Si está vacía el programa debe enviar un mensaje de error y el programa debe terminar su ejecución</p>	<p><b>Modulo Quitar( )</b></p> <p>Inicio</p> <p>    Si <math>(\text{PilaVacía}(p) = 1)</math> entonces</p> <p>        Escribir ("Pila Vacía")</p> <p>    Otro</p> <p>        <math>p \rightarrow \text{tope} \leftarrow p \rightarrow \text{tope} - 1</math></p> <p>    FinSi</p> <p>Termina</p>
Pila Vacía	<ol style="list-style-type: none"> <li>1. Verificar si el puntero de la pila vale 0</li> </ol>	<p>Devuelve 1 (verdadero) si la pila está vacía y 0 (falso) en caso contrario.</p>	<p><b>Modulo PilaVacía(*p: pila): E</b></p> <p>Inicio</p> <p>    Si <math>(p \rightarrow \text{tope} = 0)</math> entonces</p> <p>        regresa 1</p>

Operación	Algoritmo	Especificaciones	Implementación
			<p>Otro regresa 0</p> <p>FinSi</p> <p>Termina</p>
Pila Llena	<ol style="list-style-type: none"> <li>Verificar si el puntero de la pila vale el número máximo de elementos permitidos en el arreglo especificado en la declaración de la pila.</li> </ol>	<p>Devuelve 1 (verdadero) si la pila está llena y 0 (falso) en caso contrario.</p>	<p><b>Modulo PilaLlena(*p: pila): E</b></p> <p>Inicio</p> <p>Si <math>(p \rightarrow \text{tope} = \text{MAX})</math> entonces regresa 1</p> <p>Otro regresa 0</p> <p>FinSi</p> <p>Termina</p>
Inicializar	<ol style="list-style-type: none"> <li>Asignar al puntero de la pila (tope) el valor de 0.</li> </ol>	<p>Se limpia o vacía la pila, dejándola sin elementos.</p>	<p><b>Modulo Inicializar(*p : pila)</b></p> <p>Inicio</p> <p><math>p \rightarrow \text{tope} \leftarrow 0</math></p> <p>Termina</p>
Cima	<ol style="list-style-type: none"> <li>Verificar si la pila no está vacía.</li> <li>Leer el elemento situado en la posición especificado por el puntero de la pila (tope) en el arreglo.</li> </ol>	<p>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.</p> <p>Si está vacía regresa el valor de 0.</p>	<p><b>Modulo Cima(*p: pila)</b></p> <p>Inicio</p> <p>Si <math>(\text{PilaVacía}(p) = 1)</math> entonces regresa 0</p> <p>Otro regresa <math>p \rightarrow \text{datos}[p \rightarrow \text{tope}]</math></p> <p>FinSi</p> <p>Termina</p>

## COLA

Operación	Algoritmo	Especificaciones	Implementación
<b>Insertar</b>	<ol style="list-style-type: none"> <li>1. Verificar si la cola no está llena</li> <li>2. Almacenar el elemento nuevo en la posición del puntero final de la cola</li> <li>3. Incrementar en 1 el puntero del final de la cola</li> </ol>	<p>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</p>	<p><b>Modulo Insertar (dato:E)</b></p> <p><i>Inicio</i></p> <p style="padding-left: 40px;">Si(ColaLlena(c)=1) entonces</p> <p style="padding-left: 80px;">Escribir("Cola llena")</p> <p style="padding-left: 40px;">Otro</p> <p style="padding-left: 40px;">c-&gt;datos[c-&gt;final] ← dato</p> <p style="padding-left: 40px;">c-&gt;final ← c-&gt;final + 1</p> <p style="padding-left: 40px;">FinSi</p> <p><i>Termina</i></p>
<b>Quitar</b>	<ol style="list-style-type: none"> <li>1. Verificar si la cola no está vacía</li> <li>2. Leer el elemento de la posición del puntero (frente) de la cola</li> <li>3. Incrementa en 1 el frente de la cola</li> </ol>	<p>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</p>	<p><b>Modulo Quitar( )</b></p> <p><i>Inicio</i></p> <p style="padding-left: 40px;">Si (ColaVacía(c) = 1) entonces</p> <p style="padding-left: 80px;">Escribir ("Cola Vacía")</p> <p style="padding-left: 40px;">Otro</p> <p style="padding-left: 40px;">c-&gt;frente ← c-&gt;frente + 1</p> <p style="padding-left: 40px;">FinSi</p> <p><i>Termina</i></p>
<b>Cola Vacía</b>	<ol style="list-style-type: none"> <li>1. Verificar si el frente de la cola es igual al final de la cola</li> </ol>	<p>Devuelve 1 (verdadero) si la cola está vacía y 0 (falso) en caso contrario.</p>	<p><b>Modulo ColaVacía(*c: cola) : E</b></p> <p><i>Inicio</i></p> <p style="padding-left: 40px;">Si ((c-&gt;final=1 y c-&gt;frente=1) o (c-&gt;final &lt; c-&gt;frente)) entonces</p> <p style="padding-left: 80px;">Regresa 1</p> <p style="padding-left: 40px;">Otro</p> <p style="padding-left: 80px;">Regresa 0</p> <p style="padding-left: 40px;">FinSi</p> <p><i>Termina</i></p>

Operación	Algoritmo	Especificaciones	Implementación
<b>Cola Llena</b>	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.	<b>Modulo ColaLlena(*c: cola): E</b> <i>Inicio</i> <i>Si (c-&gt;final &gt; MAX) entonces</i> <i>regresa 1</i> <i>Otro</i> <i>regresa 0</i> <i>FinSi</i> <i>Termina</i>
<b>Inicializar</b>	1. Asignar tanto al puntero frente como final de la cola el valor de 1.	Se limpia o vacía la cola, dejándola sin elementos.	<b>Modulo Inicializar(*c :cola)</b> <i>Inicio</i> <i>c-&gt;frente ← 1</i> <i>c-&gt;final ← 1</i> <i>Termina</i>
<b>Frente</b>	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.	<b>Modulo Frente(*c: cola)</b> <i>Inicio</i> <i>Si (ColaVacía(c) = 1) entonces</i> <i>regresa 0</i> <i>Otro</i> <i>regresa c-&gt;datos[c-&gt;frente]</i> <i>FinSi</i> <i>Termina</i>

## LISTA SIMPLEMENTE ENLAZADA

Insertar	InsertarFin
<p><b>Módulo Insertar (valor:E, *L: Nodo)</b>  <b>Inicio</b>            *N: Nodo            N ← Asignación de memoria            Si (N ≠ Nulo) entonces              Si (L = Nulo) entonces                N-&gt;x ← valor                N-&gt;sig ← Nulo                L ← N                cabeza ← L              Otro                InsertarFin(valor,N)              FinSi            Otro              Escribir("Error en Memoria")            FinSi  <b>Termina</b></p>	<p><b>Módulo InsertarFin (valor:E, *N: Nodo)</b>  <b>Inicio</b>            *aux: Nodo            aux ← Ultimo( )            N-&gt;x ← valor            N-&gt;sig ← Nulo            aux-&gt; sig ← N  <b>Termina</b></p>
InsertarPrim	Eliminar
<p><b>Módulo InsertarPrim (valor:E, *N: Nodo)</b>  <b>Inicio</b>            N-&gt;x ← valor            N-&gt;sig ← cabeza            cabeza ← N  <b>Termina</b></p>	<p><b>Módulo Eliminar (valor:E, *L:Nodo)</b>  <b>Inicio</b>            *aux: Nodo            Si (L ≠ Nulo) entonces              aux ← Buscar(valor)              Si (aux ≠ Nulo) entonces                Si (aux = cabeza) entonces                  EliminarPrim()                Otro                  Si (aux-&gt;sig = Nulo) entonces                    EliminarFin(aux)                  Otro                    Eliminar2(aux)                  FinSi                FinSi              Otro                Escribir ("Elemento no encontrado")              FinSi            Otro              Escribir("Lista Vacía")            FinSi  <b>Termina</b></p>

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

Primero	Visualiza
<p><b>Módulo Primero( ): Nodo</b>  <b>Inicio</b>            Regresa cabeza  <b>Termina</b></p>	<p><b>Módulo Visualiza( )</b>  <b>Inicio</b>            *N: Nodo            N ← cabeza            Mientras (N ≠ Nulo)                Escribir(N-&gt;x)                N ← N-&gt;sig            FinMientras  <b>Termina</b></p>
Insertar2	Eliminar2
<p><b>Módulo Insertar2(*N:Nodo, *Nlq:Nodo, *NDer:Nodo)</b>  <b>Inicio</b>            N-&gt;sig ← NDer            Nlq-&gt;sig ← N  <b>Termina</b></p>	<p><b>Módulo Eliminar2(*N:Nodo)</b>  <b>Inicio</b>            *aux: Nodo            aux ← Anterior(N)            aux-&gt;sig ← N-&gt;sig            Liberar(N)  <b>Termina</b></p>

## LISTA DOBLEMENTE ENLAZADA

Insertar	InsertarFin
<p><b>Módulo Insertar (valor:E, *L: Nodo)</b>  <b>Inicio</b>            *N: Nodo            N ← Asignación de memoria            Si (N ≠ Nulo) entonces              Si (L = Nulo) entonces                N-&gt;x ← valor                N-&gt;sig ← Nulo                N-&gt;ant ← Nulo                L ← N                cabeza ← L                final ← L              Otro                InsertarFin(valor,N)              FinSi            Otro              Escribir("Error en Memoria")            FinSi  <b>Termina</b></p>	<p><b>Módulo InsertarFin (valor:E, *N: Nodo)</b>  <b>Inicio</b>            N-&gt;x ← valor            N-&gt;sig ← Nulo            N-&gt;ant ← final            final-&gt; sig ← N            final ← N  <b>Termina</b></p>
InsertarPrim	Eliminar
<p><b>Módulo InsertarPrim (valor:E, *N: Nodo)</b>  <b>Inicio</b>            N-&gt;x ← valor            N-&gt;ant ← Nulo            N-&gt;sig ← cabeza            cabeza-&gt; anterior ← N            cabeza ← N  <b>Termina</b></p>	<p><b>Módulo Eliminar (valor:E, *L:Nodo)</b>  <b>Inicio</b>            *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</p>

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

<i>FinMientras</i> <i>Regresa nulo</i> <b>Termina</b>	
<b>Insertar2</b>	<b>Eliminar2</b>
<b>Módulo Insertar2(*N:Nodo, *Nlzq:Nodo, *NDer:Nodo)</b> <b>Inicio</b> <i>N-&gt;sig ← NDer</i> <i>N-&gt;ant ← Nlzq</i> <i>Nlzq-&gt;sig ← N</i> <i>NDer-&gt;ant ← N</i> <b>Termina</b>	<b>Módulo Eliminar2(*N:Nodo)</b> <b>Inicio</b> <i>N-&gt;ant-&gt;sig ← N-&gt;sig</i> <i>N-&gt;sig-&gt;ant ← N-&gt;ant</i> <i>Liberar(N)</i> <b>Termina</b>

## LISTA CIRCULAR SIMPLEMENTE ENLAZADA

Insertar	InsertarFin
<p><b>Módulo Insertar (valor:E, *L: ListaC)</b>  <b>Inicio</b>            *N: ListaC            N ← Asignación de memoria            Si (N ≠ Nulo) entonces              Si (L = Nulo) entonces                N-&gt;x ← valor                N-&gt;sig ← N                L ← N                Lc ← L              Otro                InsertarFin(valor,N)              FinSi            Otro              Escribir("Error en Memoria")            FinSi  <b>Termina</b></p>	<p><b>Módulo InsertarFin (valor:E, *N: ListaC)</b>  <b>Inicio</b>            N-&gt;x ← valor            N-&gt;sig ← Lc-&gt;sig            Lc-&gt; sig ← N            Lc ← N  <b>Termina</b></p>
InsertarPrim	Eliminar
<p><b>Módulo InsertarPrim (valor:E, *N: ListaC)</b>  <b>Inicio</b>            N-&gt;x ← valor            N-&gt;sig ← Lc-&gt;sig            Lc-&gt; sig ← N  <b>Termina</b></p>	<p><b>Módulo Eliminar (valor:E, *L:ListaC)</b>  <b>Inicio</b>            *aux: ListaC            Si (L ≠ Nulo) entonces              aux ← Buscar(valor)              Si (aux ≠ Nulo) entonces                Si (aux = Lc-&gt;sig) entonces                  EliminarPrim()                Otro                  Si (aux = Lc) entonces                    EliminarFin()                  Otro                    Eliminar2(aux)                FinSi              FinSi            Otro              Escribir ("Elemento no encontrado")            FinSi</p>

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

## LISTA CIRCULAR DOBLEMENTE ENLAZADA

Insertar	InsertarFin
<p><b>Módulo Insertar (valor:E, *L: ListaCD)</b>  <b>Inicio</b>            *N: ListaCD            N ← Asignación de memoria            Si (N ≠ Nulo) entonces              Si (L = Nulo) entonces                N-&gt;x ← valor                N-&gt;sig ← N                N-&gt;ant ← N                L ← N                Lc ← L              Otro                InsertarFin(valor,N)              FinSi            Otro              Escribir("Error en Memoria")            FinSi  <b>Termina</b></p>	<p><b>Módulo InsertarFin (valor:E, *N: ListaCD)</b>  <b>Inicio</b>            N-&gt;x ← valor            N-&gt;ant ← Lc            N-&gt;sig ← Lc-&gt;sig            Lc-&gt;sig ← N            Lc ← N  <b>Termina</b></p>
InsertarPrim	Eliminar
<p><b>Módulo InsertarPrim (valor:E, *N: ListaCD)</b>  <b>Inicio</b>            N-&gt;x ← valor            N-&gt;ant ← Lc            N-&gt;sig ← Lc-&gt;sig            Lc-&gt;sig ← N  <b>Termina</b></p>	<p><b>Módulo Eliminar (valor:E, *L:ListaCD)</b>  <b>Inicio</b>            *aux: ListaCD            Si (L ≠ Nulo) entonces              aux ← Buscar(valor)              Si (aux ≠ Nulo) entonces                Si (aux = Lc-&gt;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  <b>Termina</b></p>

EliminarPrim	EliminarFin
<p><b>Módulo EliminarPrim ()</b>  <b>Inicio</b>  *aux: ListaCD  aux ← Lc-&gt;sig  Lc-&gt;sig-&gt;ant ← Lc  Lc-&gt;sig ← Lc-&gt;sig-&gt;sig  Liberar (aux)  <b>Termina</b></p>	<p><b>Módulo EliminarFin ()</b>  <b>Inicio</b>  *aux: ListaCD  aux ← Lc-&gt;ant  Lc-&gt;ant-&gt;sig ← Lc-&gt;sig  Lc-&gt;sig-&gt;ant ← Lc-&gt;ant  Liberar (Lc)  Lc ← aux  <b>Termina</b></p>

# ÁRBOL

Operación	Implementación
<p><b>Búsqueda</b></p>	<p><b>Módulo Buscar (*N: Nodo, dato:E)</b>  <b>Inicia</b>              Si (<math>N \neq \text{Nulo}</math>) entonces                  Si (<math>\text{dato} &lt; N \rightarrow \text{Num}</math>) entonces                      Buscar (<math>N \rightarrow I</math>, dato)                  Otro                      Si (<math>\text{dato} &gt; N \rightarrow \text{Num}</math>) entonces                          Buscar (<math>N \rightarrow D</math>, dato)                      Otro                          Escribir ("Elemento Encontrado")                      FinSi                  FinSi              Otro                  Escribir ("Elemento No Encontrado")              FinSi  <b>FinMódulo</b></p>
<p><b>Inserción</b></p>	<p><b>Módulo Insertar (*N: Nodo, dato:E)</b>  <b>Inicia</b>              Si (<math>N \neq \text{Nulo}</math>) entonces                  Si (<math>\text{dato} &lt; N \rightarrow \text{Num}</math>) entonces                      Insertar (<math>N \rightarrow I</math>, dato)                  Otro                      Si (<math>\text{dato} &gt; N \rightarrow \text{Num}</math>) entonces                          Insertar (<math>N \rightarrow D</math>, dato)                      Otro                          Escribir ("El Nodo ya se encuentra")                      FinSi                  FinSi              Otro                  Crea (Nuevo)                  Nuevo <math>\rightarrow I \leftarrow \text{Nulo}</math>                  Nuevo <math>\rightarrow D \leftarrow \text{Nulo}</math>                  Nuevo <math>\rightarrow \text{Num} \leftarrow \text{dato}</math></p>

Operación	Implementación
	<p style="text-align: center;"><math>N \leftarrow \text{Nuevo}</math></p> <p style="text-align: right;"><i>FinSi</i> <b>FinMódulo</b></p>
<b>Eliminación</b>	<p><b>Módulo Borrar (*N: Nodo)</b> <b>Inicia</b></p> <p style="padding-left: 2em;"><i>Si</i> (<math>N \neq \text{Nulo}</math>) entonces</p> <p style="padding-left: 4em;"><i>Si</i> (<math>\text{dato} &lt; N \rightarrow \text{Num}</math>) entonces</p> <p style="padding-left: 6em;">Borrar (<math>N \rightarrow I</math>, dato)</p> <p style="padding-left: 4em;">Otro</p> <p style="padding-left: 6em;"><i>Si</i> (<math>\text{dato} &gt; N \rightarrow \text{Num}</math>) entonces</p> <p style="padding-left: 8em;">Borrar (<math>N \rightarrow D</math>, dato)</p> <p style="padding-left: 6em;">Otro</p> <p style="padding-left: 8em;"><math>Otro \leftarrow N</math></p> <p style="padding-left: 6em;"><i>Si</i> (<math>Otro \rightarrow D = \text{Nulo}</math>) entonces</p> <p style="padding-left: 8em;"><math>N \leftarrow Otro \rightarrow I</math></p> <p style="padding-left: 6em;">Otro</p> <p style="padding-left: 8em;"><i>Si</i> (<math>Otro \rightarrow I = \text{Nulo}</math>) entonces</p> <p style="padding-left: 10em;"><math>N \leftarrow Otro \rightarrow D</math></p> <p style="padding-left: 8em;">Otro</p> <p style="padding-left: 10em;"><math>Aux \leftarrow N \rightarrow I</math></p> <p style="padding-left: 10em;"><math>Band \leftarrow 0</math></p> <p style="padding-left: 10em;"><i>Mientras</i> (<math>Aux \rightarrow D \neq \text{Nulo}</math>)</p> <p style="padding-left: 12em;"><math>Aux1 \leftarrow Aux</math></p> <p style="padding-left: 12em;"><math>Aux \leftarrow Aux \rightarrow D</math></p> <p style="padding-left: 12em;"><math>Band \leftarrow 1</math></p> <p style="padding-left: 10em;"><i>FinMientras</i></p> <p style="padding-left: 10em;"><math>N \rightarrow \text{Num} \leftarrow Aux \rightarrow \text{Num}</math></p> <p style="padding-left: 10em;"><math>Otro \leftarrow Aux</math></p> <p style="padding-left: 10em;"><i>Si</i> (<math>Band = 1</math>) entonces</p> <p style="padding-left: 12em;"><math>Aux1 \rightarrow D \leftarrow Aux \rightarrow I</math></p> <p style="padding-left: 10em;">Otro</p> <p style="padding-left: 12em;"><math>N \rightarrow I \leftarrow Aux \rightarrow I</math></p> <p style="padding-left: 10em;"><i>FinSi</i></p> <p style="padding-left: 8em;"><i>FinSi</i></p> <p style="padding-left: 6em;"><i>FinSi</i></p> <p style="padding-left: 4em;">Libera memoria (<i>Otro</i>)</p> <p style="padding-left: 2em;"><i>FinSi</i></p> <p style="padding-left: 4em;"><i>FinSi</i></p> <p style="padding-left: 2em;">Otro</p> <p style="padding-left: 4em;">Escribir ("Elemento No Encontrado")</p> <p style="padding-left: 2em;"><i>FinSi</i></p> <p><b>FinMódulo</b></p>

## GRAFO ESTÁTICO

Operación	Descripción	Implementación
<i>InicializaGrafo(G)</i>	El grafo no tiene vértices ni arcos	<p><b>Módulo IniciaGrafo(*G: Grafo)</b>  <b>Inicia</b></p> <p style="padding-left: 40px;"><i>I, J: E</i>  <i>Para I ← 1, I &lt; N, I ← I + 1</i>  <i>G → V[I] ← 0</i>  <i>Para J ← 1, J ≤ N, J ← J + 1</i>  <i>G → A[I][J] ← 0</i>  <i>FinPara</i>  <i>FinPara</i></p> <p><b>Termina</b></p>
<i>Union(u,v)</i>	Añade el arco (u, v) al grafo G. Añadir los dos vértices del arco	<p><b>Módulo AnadeArco(*G: Grafo, Arc: Arco)</b>  <b>Inicia</b></p> <p style="padding-left: 40px;"><i>AnadeVertice(G, Arc.u)</i>  <i>AnadeVertice(G, Arc.v)</i>  <i>G → A[Arc.u][Arc.v] ← 1</i></p> <p><b>Termina</b></p>
<i>BorrarArco(u,v)</i>	Elimina del grafo G el arco	<p><b>Módulo ElimArco(*G: Grafo, Arc: Arco)</b>  <b>Inicia</b></p> <p style="padding-left: 40px;"><i>G → A[Arc.u][Arc.v] ← 0</i></p> <p><b>Termina</b></p>
<i>Adyacente(u,v)</i>	Función que devuelve cero (0) si no forman un arco los vértices u, v	<p><b>Módulo PerteneceArco(*G: Grafo, Arc: Arco)</b>  <b>Inicia</b></p> <p style="padding-left: 40px;"><i>Sw: E</i>  <i>Sw ← PerteneceVertice(G, Arc.u) y</i>  <i>PerteneceVertice(G, Arc.v)</i>  <i>Si (Sw = 1) entonces</i>  <i>Regresa G → A[Arc.u][Arc.v]</i>  <i>Otro</i>  <i>Regresa 0</i>  <i>FinSi</i></p> <p><b>Termina</b></p>

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

## GRAFOS DINÁMICOS

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

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

Operación	Implementación
	<pre> 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-&gt;el.Ady, Dato1) </pre> <p><b>Termina</b></p>
BorraArco	<p><b>Módulo BorraArco(*Primero: ListaG, arc: Arco)</b></p> <p><b>Inicia</b></p> <pre> 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-&gt;el.Ady, Dato1) </pre> <p><b>Termina</b></p>
AñadeVértice	<p><b>Módulo AñadeVertice(*Primero: ListaG, dato: ItemAdy)</b></p> <p><b>Inicia</b></p> <pre> *Nuevo: ListaG Si (PertenceVertice(Primero,dato) ≠ 1) entonces     Nuevo ← Asignar memoria     Nuevo-&gt;el ← dato     Nuevo-&gt;sig ← Primero     Primero ← Nuevo </pre> <p><i>FinSi</i></p> <p><b>Termina</b></p>
BorraVértice	<p><b>Módulo BorraVertice(*Primero: ListaG, dato:ItemG)</b></p> <p><b>Inicia</b></p> <pre> *ptr, *ant: ListaG Dato1: ItemAdy EncuentraPosGrafo(Primero, dato, ant, ptr) Si (ptr ≠ nulo) entonces     Si (ptr = primero) entonces         Primero ← ptr-&gt;sig     Otro         ant-&gt;sig ← ptr-&gt;sig </pre> <p><i>FinSi</i></p>

Operación	Implementación
	<pre> BorraListaAdy(ptr-&gt;el.Ady) Liberar (ptr) ptr ← primero dato1.v ← dato.v Mientras (ptr ≠ nulo)     BorraConjuntoAdy(ptr-&gt;el.Ady,dato1)     ptr ← ptr-&gt;sig FinMientras FinSi <b>Termina</b> </pre>
PerteneceVertice	<pre> <b>Módulo PerteneceVertice(*Primero: ListaG, dato: ItemAdy)</b> <b>Inicia</b>     *ptr: ListaG     Para ptr←Primero, ptr ≠ nulo, ptr ← ptr-&gt;sig         Si (ptr-&gt;el.v = dato.v) entonces             Regresa 1         FinSi     FinPara     Regresa 0 <b>Termina</b> </pre>
PerteneceArco	<pre> <b>Módulo PerteneceArco (*Primero: ListaG, dato: ItemAdy)</b> <b>Inicia</b>     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-&gt;el.Ady, Dato1,Ant1,Pos1)         Si (Pos1 ≠ Nulo) entonces             Regresa 1         Otro             Regresa 0         FinSi     FinSi     Regresa 0 <b>Termina</b> </pre>

