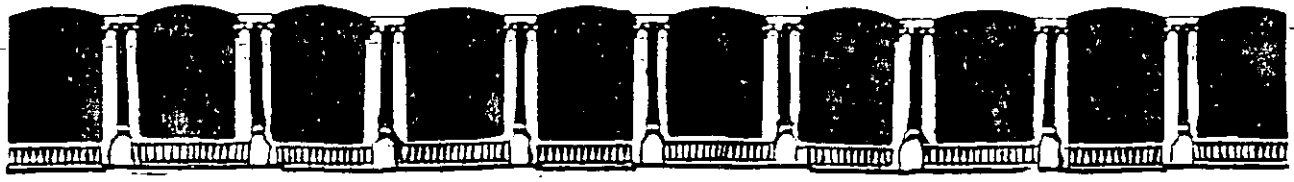


FACULTAD DE INGENIERIA U.N.A.M.
DIVISION DE EDUCACION CONTINUA

A LOS ASISTENTES A LOS CURSOS DE LA DIVISION DE EDUCACION CONTINUA

Las autoridades de la Facultad de Ingeniería, por conducto del Jefe de la División de Educación Continua, otorgan una constancia de asistencia a quienes cumplan con los requisitos establecidos para cada curso.

El control de asistencia se llevará a cabo a través de la persona que le entregó las notas. Las inasistencias serán computadas por las autoridades de la División, con el fin de entregarle constancia sólamente a los alumnos que tengan un mínimo del 80% de asistencias.

Pedimos a los asistentes recoger su constancia el día de la clausura. Estas se retendrán por el período de un año, pasado este tiempo la DECFI no se hará responsable de este documento.

Se recomienda a los asistentes participar activamente con sus ideas y experiencias, pues los cursos que ofrece la División están planeados para que los profesores expongan una tésis, pero sobre todo, para que coordinen las opiniones de todos los interesados, constituyendo verdaderos seminarios.

Es muy importante que todos los asistentes llenen y entreguen su hoja de inscripción al inicio del curso, información que servirá para integrar un directorio de asistentes, que se entregará oportunamente.

Con el objeto de mejorar los servicios que la División de Educación Continua ofrece, al final del curso deberán entregar la evaluación a través de un cuestionario diseñado para emitir juicios anónimos.

Se recomienda llenar dicha evaluación conforme los profesores impartan sus clases, a efecto de no llenar en la última sesión las evaluaciones y con ésto sean más fehacientes sus apreciaciones.

¡ G R A C I A S !

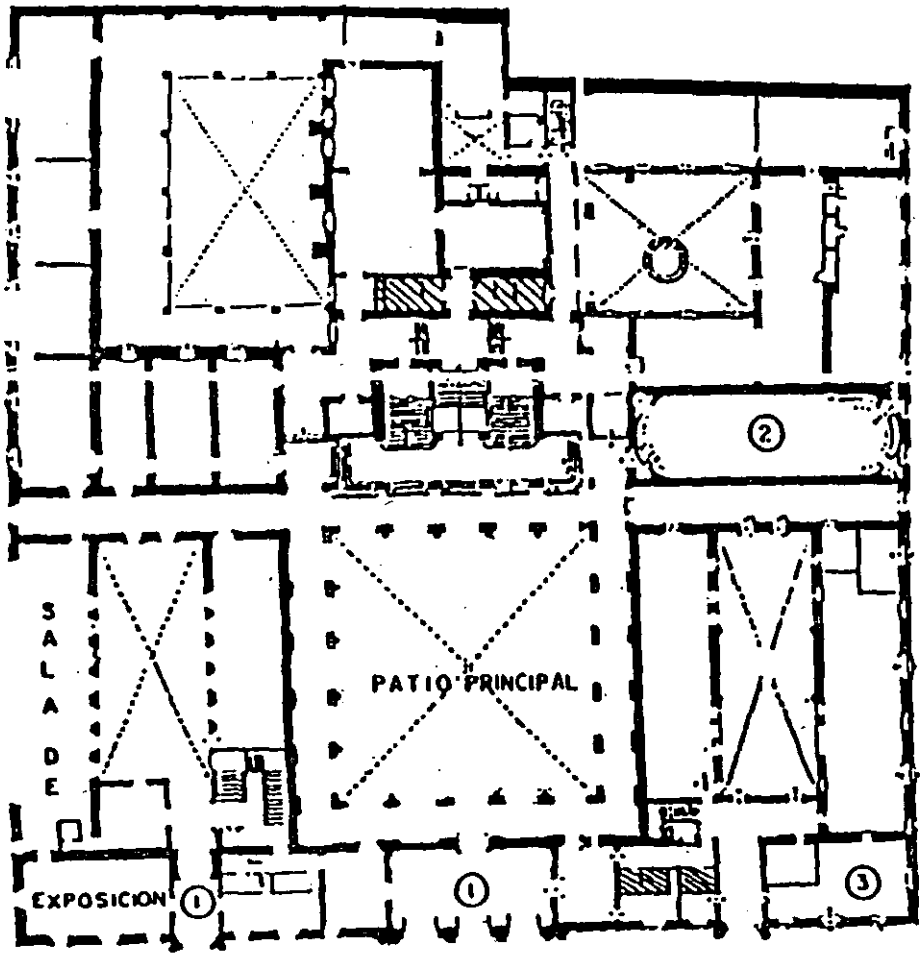
Palacio de Minería Calle de Tacuba 5 Primer piso Deleg. Cuauhtémoc 06000 México, D.F. APDO. Postal M-2285
Teléfonos: 512-8955 512-5121 521-7335 521-1987 Fax 510-0573 521-4020 AL 26

[illegible]

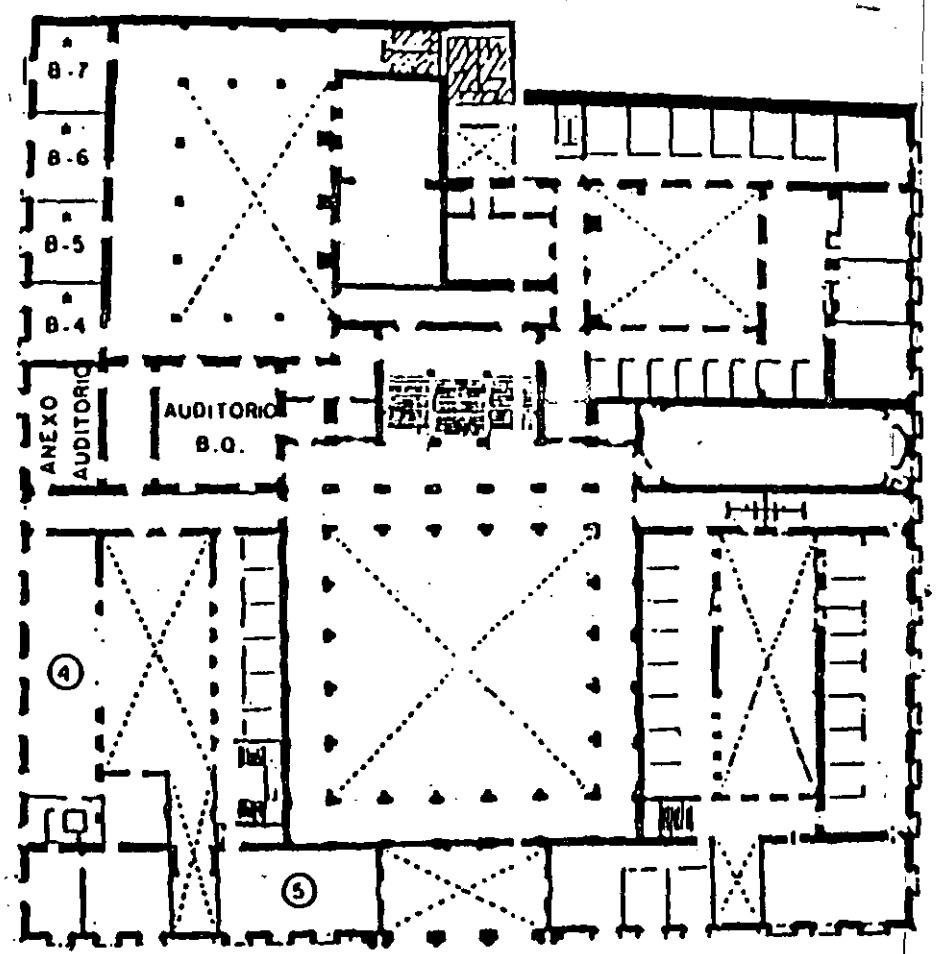
[illegible] la pesq[illegible] [illegible] utor [illegible]

[illegible] Tlalcoíla [illegible] dé un [illegible]

PALACIO DE MINERIA

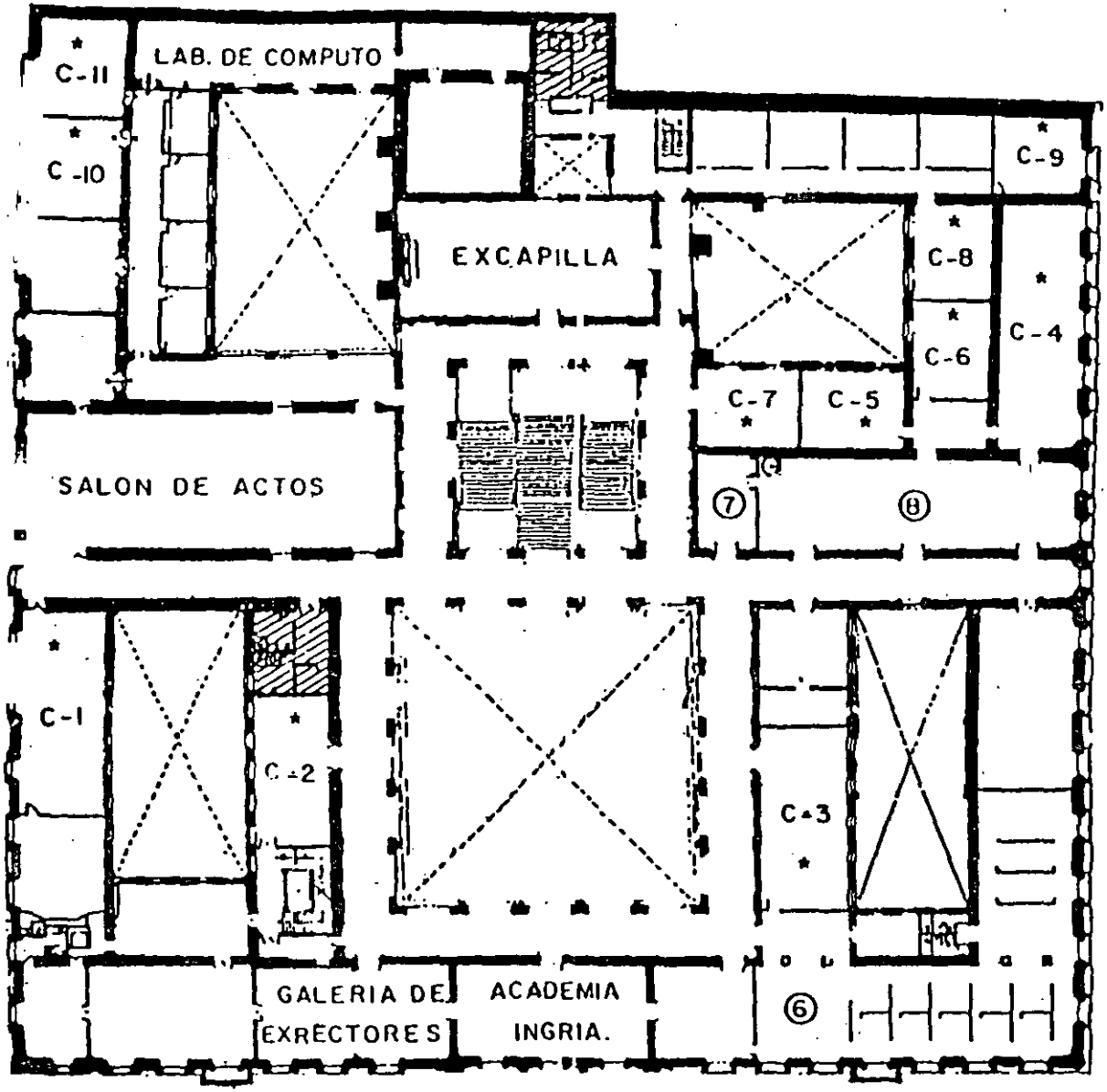


PLANTA BAJA



MEZZANINNE

DIVISION DE EDUCACION CONTINUA
FACULTAD DE INGENIERIA U.N.A.M.
CURSOS ABIERTOS



1er. PISO

GUIA DE LOCALIZACION

1 – ACCESO

2 – BIBLIOTECA HISTORICA

3 – LIBRERIA U N A M

4 – CENTRO DE INFORMACION Y DOCUMENTACION "ING. BRUNO MASCANZONI"

5 – PROGRAMA DE APOYO A LA TITULACION

* AULAS

6 – OFICINAS GENERALES

7 – ENTREGA DE MATERIAL Y CONTROL DE ASISTENCIA.

8 – SALA DE DESCANSO

▨ SANITARIOS

FACULTAD DE INGENIERIA U.N.A.M.
DIVISION DE EDUCACION CONTINUA

CURSOS INSTITUCIONALES

TELEINDUSTRIA ERICSSON, S.A. DE C.V.

LENGUAJE "C" PARTE II

Del 21 de Octubre al 9 de Noviembre

Ing. Jessica Briseño C.

Palacio de Minería

1994

LISTAS LINEALES

LISTA

Las listas son estructuras de datos que dan mayor flexibilidad de programación a los usuarios, con un ahorro considerable de memoria por las operaciones que pueden practicarse sobre ella.

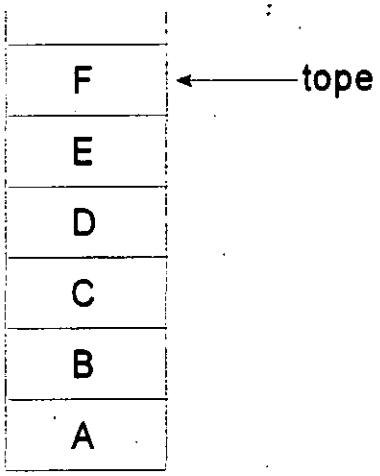
Una lista es una estructura de datos que tiene un número variable de nodos.

Una lista lineal, es una lista cuyos nodos están ordenados por un solo criterio, en donde el último y el primer nodo no tienen sucesor y antecesor respectivamente.

PILA

Una pila o stack es una estructura de datos lineal, en la cual las operaciones se realizan por uno de los extremos de la lista.

Una pila se puede representar mediante la siguiente figura:

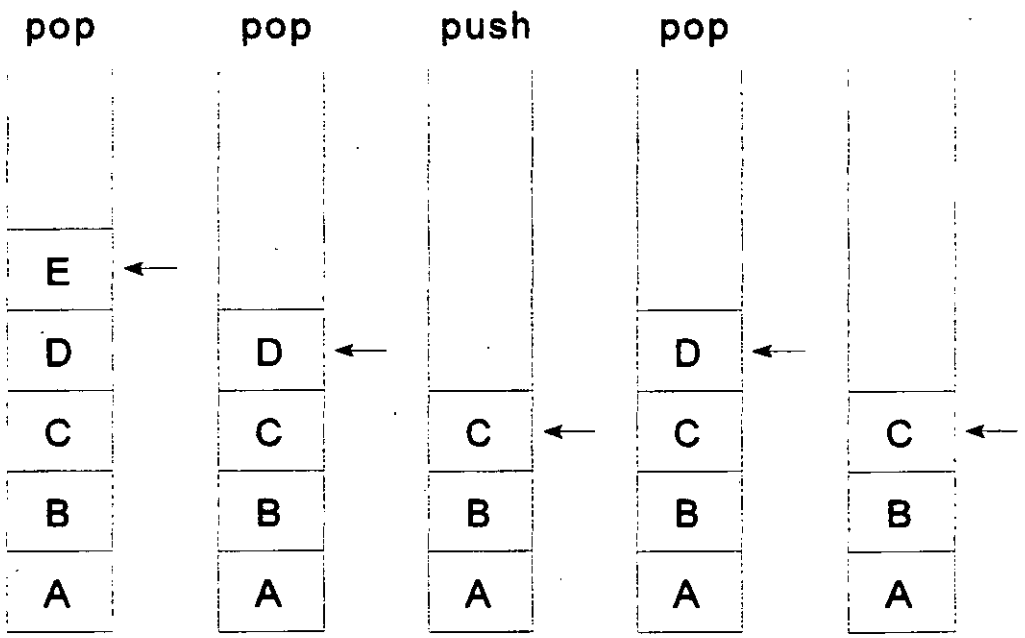


Uno de los extremos de la pila se designa como el tope de la misma y es por donde se colocan nuevos elementos o se retiran.

En el caso de que se agreguen nuevos elementos a la pila, el tope se moverá hacia arriba para indicar al último elemento en entrar a la pila.

En el caso de que se retire un elemento de la pila el tope se mueve hacia abajo, para indicar hacia el nuevo elemento que se encuentra en el extremo de la pila.

En la siguiente figura se muestra el comportamiento de la pila al insertar y borrar elementos de ella.



Por la forma en que se agregan y retiran elementos de la pila, el método ha sido llamado LIFO (las input first output). esto significa que solamente puede ser retirado de la pila el último elemento agregado.

Las operaciones que se llevan a cabo sobre una pila se conocen como **push** (para insertar) y **pop** (para borrar un elemento).

Cuando la pila contiene un solo elemento y se lleva a cabo una operación de pop, la pila resultante no contiene elementos y se llama **pila vacía**.

Aunque la operación push es aplicable teóricamente en cualquier momento, no ocurre lo mismo con la operación pop, la cual no puede aplicarse a una pila vacía.

Representación de pilas en C

Antes de programar la solución de un problema que use una pila, debe decidirse como representarla mediante las estructuras de datos existentes en nuestro lenguaje de programación.

Como se verá, hay varias maneras de representar una pila en el lenguaje C. Consideremos ahora la más sencilla: un *arreglo*.

Sin embargo, una pila y un arreglo son dos cosas diferentes. El número de elementos de una arreglo es fijo y se asigna en la definición de este. Por otra parte, una pila es un objeto dinámico cuyo tamaño cambia constantemente mediante las operaciones push y pop.

Si se quiere implementar una pila con un arreglo se deberá definir un arreglo lo bastante grande para admitir el tamaño máximo de la pila. Es necesario otro elemento para guarde la posición del elemento tope de la pila.

Por lo tanto puede declararse una pila, como una estructura que contiene dos elementos: una arreglo para guardar los elementos de la pila y un entero para guardar la posición del elemento tope de la pila:

```
# define STACKSIZE      100

typedef     struct {
                int     tope;
                int     elementos[STACKSIZE];
}       STACK;
```

Para definir una pila:

```
STACK       s;
```

Las funciones push y pop se definirían de la siguiente forma:

```
void push(STACK *s, int dato) {

if ( s->tope == STACKSIZE )
        printf("Pila llena");
else
        s->elementos[s->tope++] = dato;
}


int  pop(STACK *s) {

if ( s->tope == 0 )
        printf("Pila vacía");
else
        return s->elementos[s->tope--];
}
```

Para que no existiera restricción en cuanto al tipo de elementos en la pila se podría definir de la siguiente forma:

```
# define STACKSIZE      100

typedef     union {
        int     intEle;
        float   floatEle;
        char    charEle;
} ELEMENTO;


typedef     struct {
            int     tope;
            ELEMENTO  elementos[STACKSIZE];
}       STACK;
```

Pilas dinámicas

Una de las características principales de una pila es que su tamaño es dinámico, es decir su tamaño no esta previamente definido.

Para definir una pila dinámica utilizaremos la siguiente estructura:

```
typedef     struct s {
            int     dato;
            struct s *liga;
} ELEMENTO;

typedef  ELEMENTO      *STACK;
```

De esta forma se puede definir una pila como:

```
STACK       pila= NULL;
```

Inicialmente la pila esta vacía y el único espacio en memoria ocupado es el del apuntador.

Para implementar las funciones push y pop se deben considerar los siguientes puntos:

- La pila esta vacía cuando la variable de tipo STACK tiene como valor NULL.
- Cuando se lleva a cabo la operación pop, se deberá liberar la memoria ocupada por el elemento saliente.
- Cuando se lleva a cabo la operación push se deberá alojar memoria para el elemento que se va a colocar en el tope de la pila.
- Idealmente no existe límite en cuanto al número de elementos que pueden entrar a la pila.

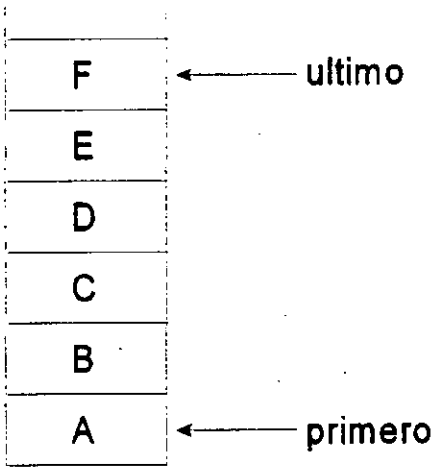
La implementación de estas funciones sería:

```
void push(STACK *s, int dato) {

ELEMENTO  *aux;

aux = (ELEMENTO *)malloc(sizeof(ELEMENTO));
aux->dato = dato;
aux->liga = *s;
*s = aux;
}


int  pop(STACK *s) {

int          dato;
ELEMENTO  *aux;

if ( *s == NULL )
        printf("Pila vacía");
else {
        dato = (*s)->dato;
        aux = *s;
        *s = (*s)->liga;
        free(aux);
        return dato;
}
}
```

COLA

Una cola es una estructura de datos lineal, en la cual la operación de inserción de un elemento se realiza por uno de los extremos de la lista y la extracción por el otro.

Una cola se puede representar mediante la siguiente figura:

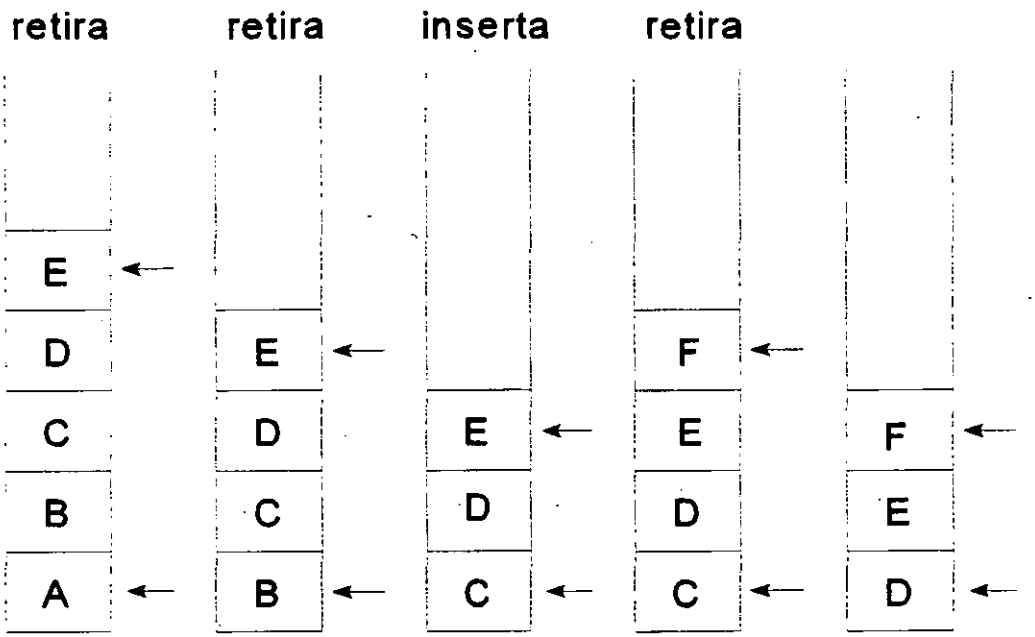


Uno de los extremos de la cola se designa como el último de la misma y es por donde se colocan nuevos elementos.

En el caso de que se agreguen nuevos elementos a la cola, este elemento será el último de la cola.

En el caso de que se retire un elemento de la cola, este elemento será el que entro antes que todos los elementos actuales de la cola. Al retirar al primer elemento de la cola, se mueven hacia abajo los demás elementos.

En la siguiente figura se muestra el comportamiento de la cola al insertar y borrar elementos de ella.



Por la forma en que se agregan y retiran elementos de la cola, el método ha sido llamado FIFO (first input first output). esto significa que solamente puede ser retirado de la cola el primer elemento agregado.

Las operaciones que se llevan a cabo sobre una cola se conocen como **insertar** y **retirar**.

Al igual que con la pila, las formas de representar una cola son muchas; sin embargo debido a que una de las características principales de una cola es que su tamaño es dinámico, consideraremos únicamente esta forma de implementación.

Para definir una cola dinámica utilizaremos la misma estructura que para la pila:

```
typedef     struct s {
            int     dato;
            struct s *liga;
} ELEMENTO;

typedef  ELEMENTO       *COLA;
```

De esta forma se puede definir una cola como:

```
COLA   cola= NULL;
```

Inicialmente la cola esta vacía y el único espacio en memoria ocupado es el del apuntador.

Para implementar las funciones insertar y retirar se deben considerar los siguientes puntos:

- La cola esta vacía cuando la variable de tipo COLA tiene como valor NULL.

- Cuando se lleva a cabo la operación retirar, se deberá liberar la memoria ocupada por el elemento saliente.

- Cuando se lleva a cabo la operación insertar se deberá alojar memoria para el elemento que se va a colocar en uno de los extremos de la cola.

- Idealmente no existe límite en cuanto al número de elementos que pueden entrar a la cola.

La implementación de las funciones para manejo de colas se muestra a continuación:

```
void inserta(COLA *s, int dato) {

ELEMENTO  *aux;

aux = (ELEMENTO *)malloc(sizeof(ELEMENTO));
aux->dato = dato;
aux->liga = *s;
*s = aux;
}


int  retira(COLA *s) {

int          dato;
ELEMENTO  *aux;

if ( *s == NULL )
        printf("Cola vacía");
else {
        aux = *s;
        while ( aux->liga != NULL )
                aux = aux->liga;
        dato = aux->dato;
        if ( aux == *s)
                *s = NULL;
        free(aux);
        return dato;
}
}
```

LABORATORIO

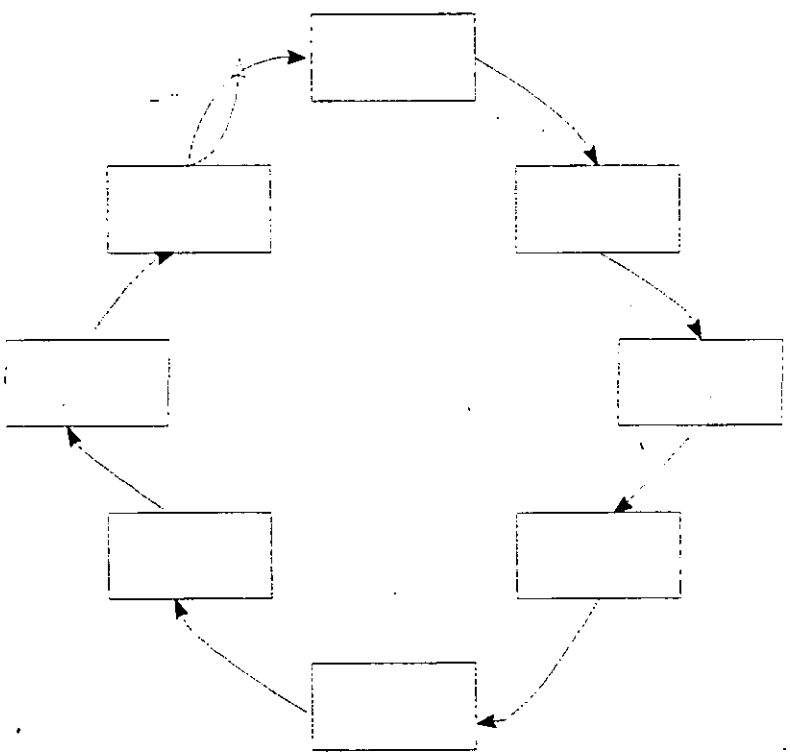
Escriba un programa que evalue una expresión en notación polaca. El programa deberá utilizar un stack para la evaluación. Las expresiones a evaluar tendrán operandos de un solo digito.

LISTA CIRCULAR

Una lista circular es aquella estructura de datos que tiene como característica fundamental un orden en el que, al último elemento le sigue el primero.

Las operaciones que se definen sobre la lista circular son las de insertar y extraer y su comportamiento depende de su manejo puede ser como cola o como pila.

Una cola se puede representar mediante la siguiente figura:



La implementación de una lista circular, considerando la representación como la de una cola es la siguiente:

```
typedef     struct s {
            int     dato;
            struct s *liga;
} ELEMENTO;

typedef  ELEMENTO      *COLA;
```

De esta forma se puede definir una cola como:

```
COLA   cola= NULL;
```

Para implementar las funciones insertar y retirar se deben considerar los siguientes puntos:

- La cola esta vacía cuando la variable de tipo COLA tiene como valor NULL.
- Cuando la cola tiene un solo elemento este apunta así mismo.

La implementación de las funciones se presenta a continuación:

```
void inserta(COLA *s, int dato) {

ELEMENTO  *aux;

aux = (ELEMENTO *)malloc(sizeof(ELEMENTO));
aux->dato = dato;
if ( *s == NULL)
        aux->liga = aux;
else {
        aux->liga = *s;
        while ( (*s)->liga != aux->liga)
                *s = (*s)->liga;
        (*s)->liga = aux;
}
*s = aux;
}
```

```
int  extrae(COLA *s) {
int          dato;
ELEMENTO  *aux;

if ( *s == NULL )
        printf("Cola vacía");
else {
        aux = *s;
        if ( aux->liga == *s) {
                dato = aux->dato;
                free(aux);
                *s = NULL;
        } else {
                while ( aux->liga->liga != *s )
                        aux = aux->liga;
                dato = aux->liga->dato;
                free(aux->liga);
                aux->liga = *s;
        }
        return dato;
}
}
```

LABORATORIO

Escriba un programa que implemente el juego de José.

LISTAS NO LINEALES
ARBOLES

En el capítulo anterior se expuso que una lista lineal es una estructura de datos que expresa las relaciones entre los nodos por un solo criterio o en una sola dimensión.

Las listas no lineales son estructuras de datos más complejas, cuyas relaciones entre sus nodos son en más de una dimensión.

Una de las listas no lineales más utilizadas, principalmente en la implementación de sistemas operativos y DBMS's, son los árboles.

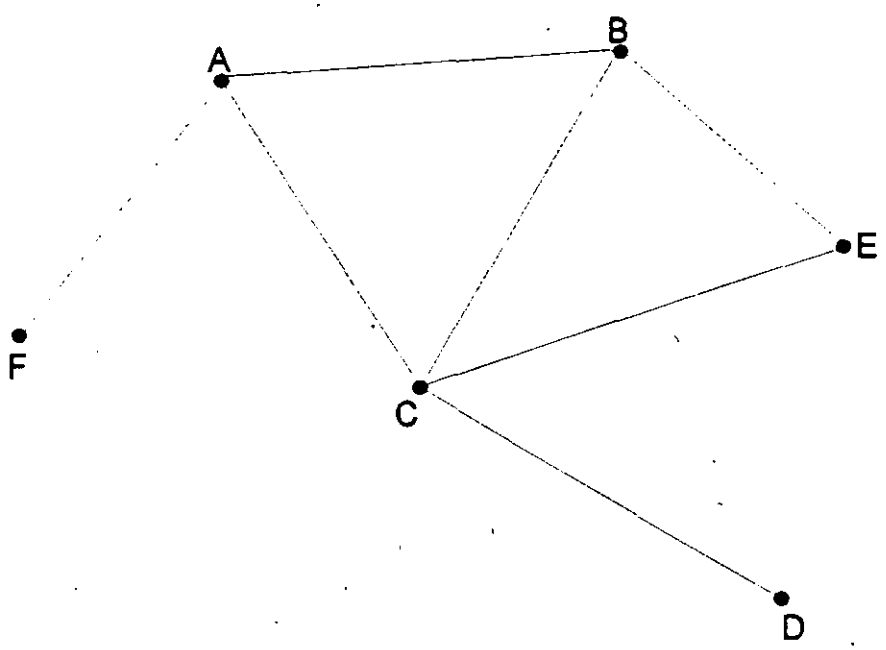
ARBOLES

Para poder definir el concepto de un árbol, tendremos que hacer referencia primero a lo que es una gráfica.

Una gráfica es un conjunto de pares ordenados:

$$G = \{ (a,b), (a,c), (c,e), (b,e), (c,d), (a,f) \}$$

representada de la siguiente forma:



Arco dirigido

Si en un par ordenado (a,b) es importante considerar que a es el nodo inicial y b el nodo final, estaremos hablando de un arco dirigido.

Grado externo de un nodo

El grado externo de un nodo u es el número de arcos que salen de él.

Grado interno de un nodo

El grado externo de un nodo u es el número de arcos que llegan a él.

Trayectoria

Si en una gráfica con arcos dirigidos ciertos pares ordenados pueden ser colocados en una secuencia de la forma:

$$(a_1, a_2), (a_2, a_3), (a_3, a_4), \ldots, (a_{n-1}, a_n)$$

el conjunto de arcos es llamado una trayectoria desde a_1 hasta a_n. Si $a_1 = a_n$, la trayectoria es un ciclo.

Cuando los arcos de la secuencia son distintos, la trayectoria es simple. Si los arcos son distintos y contienen a todos los nodos de la gráfica, la trayectoria es hamiltoniana.

La longitud de una trayectoria es el número de arcos que la componen.

Lazo o loop

Un arco que une un vértice consigo mismo se llama lazo. La dirección de una lazo no tiene ningún significado.

Definición de Arbol

Con los conceptos anteriores podemos definir un árbol:

Un árbol es una gráfica en la que:

- El número de nodos es igual al número de arcos más uno.
- Todos los nodos son de grado interno uno, excepto un nodo llamado raíz, de grado cero.
- No hay ciclos.
- Cualquier trayectoria es simple.
- Entre cualquier par de nodos solo hay una trayectoria.
- Cualquier arco es un arco de desconección.

En la terminología que se emplea para el estudio de los árboles encontraremos entre otros, los términos siguientes:

Se define como grado o grado externo de un nodo al número de sus subárboles.

Una hoja o nodo terminal es un nodo de grado cero.

Un nodo ramal es un nodo de grado mayor de cero.

El nivel de un nodo es el nivel de su antecesor directo más uno. El nivel de la raíz es uno.

Es frecuente que los nodos de un árbol reciban nombres, tales como: el nodo a es padre de b, c, d, si existe un arco de a a b, uno de a a c y otro de a a d.

ARBOLES BINARIOS

Los árboles binarios son aquellos cuyos nodos tienen un grado externo menor o igual a dos.

Los árboles binarios tienen muchas aplicaciones en sistemas operativos y bases de datos.

Las operaciones que se definen para un árbol binario son: inserción y recorrido.

El recorrido de un árbol binario es el procedimiento de visitar cada uno de sus nodos, con el objeto de sistematizar la recuperación de la información almacenada.

Una de las formas más simples de recorrer un árbol es de la raíz hacia los nodos hoja (top-down).

El recorrido top-down de un árbol binario puede ser de tres formas diferentes:

- preorden
- inorden
- postorden

En el recorrido preorden:

- se visita la raíz
- se recorre el subárbol izquierdo
- se recorre el subárbol derecho

En el recorrido inorden:

- se recorre el subárbol izquierdo
- se visita la raíz
- se recorre el subárbol derecho

En el recorrido postorden:

- se recorre el subárbol izquierdo
- se recorre el subárbol derecho
- se visita la raíz

La forma de implementar los algoritmos de recorrido es en forma recursiva y no recursiva; sin embargo, la forma recursiva es mucho más entendible, por lo que es la que se presentará.

Para ello consideremos la siguiente estructura:

```
typedef struct x {
                    int             dato;
                    struct x  *ligaIzq,
                              *ligaDer;
}       ELEMENTO;

typedef       ELEMENTO  *ARBOL;
```

Para definir una variable:

```
ARBOL       arbol=NULL;
```

Las funciones de recorrido se muestran a continuación:

```
int preOrden(ARBOL  a) {

if ( a != NULL ) {
        printf("%d ", a->dato);
        preOrden(a->ligaIzq);
        preOrden(a->ligaDer);
}
}
```

```
int inOrden(ARBOL  a) {

if ( a != NULL ) {
        inOrden(a->ligalzq);
        printf("%d ", a->dato);
        inOrden(a->ligaDer);
}
}


int postOrden(ARBOL  a) {

if ( a != NULL ) {
        postOrden(a->ligalzq);
        postOrden(a->ligaDer);
        printf("%d ", a->dato);
}
}
```

LABORATORIO

Escriba un programa que ordène una secuencia de números utilizando un árbol binario.

FACULTAD DE INGENIERIA U.N.A.M.
DIVISION DE EDUCACION CONTINUA

CURSOS INSTITUCIONALES

LENGUAJE "C" PARTE II

Del 21 de Octubre al 7 de Noviembre

TELEINDUSTRIA ERICSSON , S.A. DE C.V.

METODOS DE ORDENAMIENTO E INSERCION

Ing. Jessica Briseño Cortés

México, D.F.

1994

METODOS DE ORDENAMIENTO E INSERCION

METODOS DE ORDENAMIENTO

El ordenamiento es una de las operaciones más importantes que se practica sobre una estructura de datos.

Ordenar una estructura de datos es establecer un orden de precedencia entre los elementos de la estructura, de acuerdo a uno o más campos que se seleccionen para tal fin.

Es frecuente encontrar operaciones de ordenamiento como parte de los procesos para el manejo de datos.

Ordenar un conjunto de datos puede parecer una operación trivial, pero en realidad es una operación costosa que deberá realizarse solamente cuando sea estrictamente necesario y en este caso seleccionar el método apropiado.

Consideraciones para la selección del método de ordenamiento

Para seleccionar un método de ordenamiento en una cierta situación, es aconsejable considerar los siguiente:

1. El tipo de memoria en la que se encuentran los datos. Esta puede ser de acceso directo y de alta velocidad, de acceso directo y de mediana velocidad, y. de acceso secuencial.

2. Las características del sistema operativo para el manejo de archivos y de la memoria.

3. Los tiempos de acceso a los dispositivos.

4. La cantidad, el tipo y la distribución inicial de los datos.

5. La eficiencia del método. Para establecer la eficiencia de un método de ordenamiento, basta con determinar el número promedio de comparaciones y de intercambios, así como la cantidad de memoria que requiere.

Para tener una idea del comportamiento de los algoritmos de ordenamiento, los mejores métodos realizan un número de comparaciones proporcional a $n\log_2 n$, donde n es el número de elementos a ordenar.

Ordenamientos internos

Los métodos de ordenamiento que se utilizan para un conjunto de datos almacenados en una memoria de acceso directo de alta velocidad, son llamados métodos de ordenamiento interno.

Los métodos de ordenamiento interno se encuentran clasificados en métodos de selección, intercambio, inserción, distribución e intercalación de acuerdo al principio en el que se basan.

A continuación se listan los algoritmos más importantes en cada caso:

Métodos por selección:	directa repetitiva torneo heap
Métodos por intercambio:	burbuja transposición de pares y nones embudo quick
Métodos por inserción:	directa binaria de doble entrada shell

Métodos por distribución: cubetas

Métodos por intercalación: merge

Métodos por Selección: selección directa

Los métodos por selección como su nombre lo indica, seleccionan del conjunto de datos, según el criterio que se siga, al mayor o al menor de los datos y lo excluye para proceder sobre los restantes de forma similar.

El método de Selección Directa consiste en seleccionar del conjunto de datos el elemento más pequeño en valor y excluirlo del conjunto de datos para repetir el procedimiento sobre los restantes.

El número de comparaciones que este algoritmo realiza es:

$$(n_2 - n)/2$$

```
/*      Algoritmo de ordenamiento: Método de selección directa

        Intercambia el elemento iesimo de un arreglo de N
        elementos (0 < i < N) con el elemento menor del arreglo.

*/

#include <stdio.h>
#define swap(a,b)   {int t; t=a; a=b; b=t;}
#define maxN        100

void seleccion(int a[], int N) {

        int i, j, min;

        for(i=0; i<N-1; i++) {
                min = i;
                for(j= i+1; j < N; j++)
                        if(a[j] < a[min])
                                min=j;
                swap(a[min],a[i]);
        }
}


main() {
        int N = 0, i, a[maxN];

        while(scanf("%d",&a[N]) == 1)
                N++;
        seleccion(a,N);
        for(i=0; i < N; i++)
                printf("%d ",a[i]);
}
```

Métodos por intercambio: burbuja

Los métodos por intercambio transponen o intercambian sistemáticamente pares de datos que se encuentran fuera de orden hasta que dejen de existir.

El nombre de burbuja se debe a la manera como los datos se mueven dentro del conjunto, aparentando ser burbujas en el agua subiendo a la superficie.

El algoritmo se inicia comparando las llaves n y n-1, y las intercambia si n<n-1, compara después n-1 y n-2 y las intercambia si n-1<n-2. Este procedimiento se practica hasta comparar los datos 1 y 2 y, cuando esto sucede, el dato menor ha alcanzado su posición final.

```
/*      Algoritmo de ordenamiento: Método de la burbuja

        En varias pasadas sobre un arreglo, se intercambian
        elementos adyacentes de ser necesario.
        Después de varias pasadas no se necesitan más intercambios,
        el arreglo queda ordenado.

*/

#include <stdio.h>
#define swap(a,b)   {int t; t=a; a=b; b=t;}
#define maxN        100

void bubble(int a[], int N) {

        int i, j;

        for(i=N-1; i>=0; i--)
                for(j= 1; j <= i; j++)
                        if(a[j-1] > a[j])
                                swap(a[min],a[i]);
}

main() {
        int N = 0, i, a[maxN];

        while(scanf("%d",&a[N]) == 1)
                N++;
        bubble(a,N);
        for(i=0; i < N; i++)
                printf("%d ",a[i]);
}
```

Métodos por Intercambio: quickSort

El algoritmo de quicksort es un procedimiento recursivo en el que se intercambian los datos para colocarlos en orden con respecto a uno de ellos, llamado pivote, de tal manera que a la derecha del pivote quedan los elementos mayores a el y a la izquierda los menores.

Este proceso se repite sobre la lista de datos a la derecha del pivote y sobre la lista de datos a la izquierda del pivote.

El algoritmo es muy eficiente para llaves que se encuentran aleatoriamente distribuidas.

```
/*      Algoritmo de ordenamiento: Método de QuickSort

        Este método consiste en dividir el arreglo a ordenar
        en dos subarreglos formados a partir de un elemento llamado
        pivote.

*/

#include <stdio.h>
#define swap(a,b)   {int t; t=a; a=b; b=t;}
#define maxN        100

void quickSort(int a[], int l, int r) {

        int i, j, v;

        if (r>1) {
                v=a[r];
                i=l-1;
                j=r;
                for(;;) {
                        while(a[++i] < v)
                                ;
                        while(a[--j] > v)
                                ;
                        if (i >= j)
                                break;
                        swap(a[i],a[j]);
                }
                swap(a[i],a[r]);
                quickSort(a,l,i-1);
                quickSort(a,i+1,r);
        }
}
```

```
main() {

        int N = 0,
        i,
        a[maxN+1];

        while(scanf("%d",&a[N]) == 1)
                N++;
        quickSort(a,1,N);
        for(i=0; i < N; i++)
                printf("%d ",a[i]);

}
```

Métodos por Inserción: Inserción Directa

Los métodos por inserción suponen que el conjunto de llaves se encuentra ordenado. Para una llave K que se desea agregar al conjunto, se determina el lugar que debe ocupar y se mueven los datos una posición para insertarlo en su posición correcta.

```
/*      Algoritmo de ordenamiento: Método de Insercion Directa

        Los elementos del arreglo se ordenan insertando el elemento a[i] en la posición
        adecuada dentro del conjunto ordenado de elementos a[1]...a[i-1].

*/


#include <stdio.h>
#define maxN        100


void insercion(int a[], int N, int x) {

        int i, j, v;

        for(i=0; a[i]<x && i < N; i++)

        for(j = N; j > i; j--)
                a[j] = a[j-1];
        a[i] = x;
}
```

```
main() {

        int N = 0,
        dato,
        a[maxN+1];


        while(scanf("%d",&dato) == 1)
                insercion(a, ++N, dato);

        for(i=0; i < N; i++)
                printf("%d ",a[i]);

}
```