



ESTRUCTURA DE DATOS

TEMA 2. COLAS

Presenta: Mtro. David Martínez Torres
Universidad Tecnológica de la Mixteca
Instituto de Computación

Oficina No. 37
dtorres@gs.utm.mx

Contenido

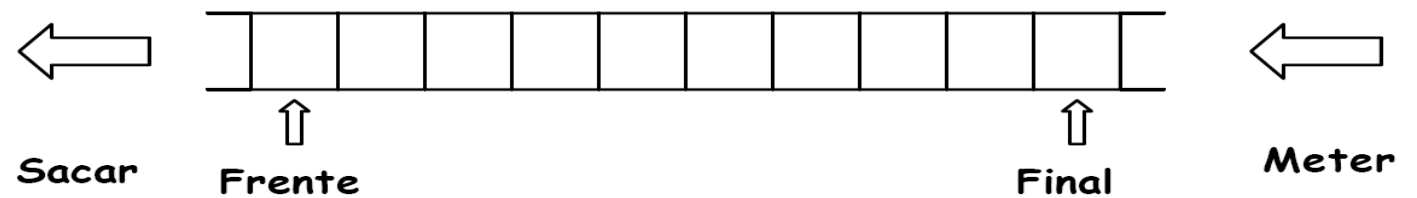
1. Definición y operaciones
2. Implementación estática
3. Implementación dinámica
4. Colas de prioridad
5. Casos de estudio

1. Definición y operaciones

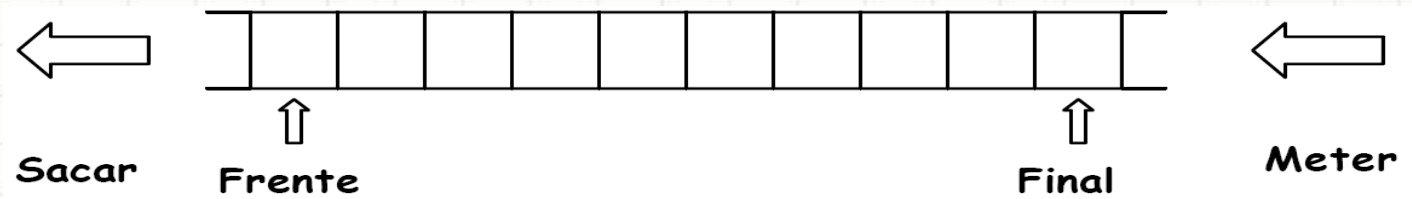
La **cola** es una colección ordenada de elementos de la que se pueden borrar elementos en un extremo (llamado el frente de la cola) e insertarlos por el otro (llamado el final de la cola).



El primer elemento insertado en la cola, es el primero en ser eliminado; razón por la que a las colas se les conoce como una estructura de datos **FIFO** (por first-in, first-out)



1. Definición y operaciones



- Insertar(enqueue) un elemento a la cola
`insertar(&cola, elemento);`
- Eliminar(dequeue) un elemento de la cola
`elemento= borrar(&cola);`
- Determinar si la cola esta vacía
`bandera=vacia(cola);`
- Determinar si la cola esta llena
`bandera=llena(cola);`

Ejemplificar:

1. `insertar(cola,A);`
2. `insertar(cola,B);`
3. `insertar(cola,C);`
4. `eliminar(cola);`
5. `eliminar(cola);`
6. `insertar(cola,D);`
7. `eliminar(cola);`
8. `eliminar(cola);`
9. `insertar(cola,E);`

1. Definición y operaciones: Aplicaciones con colas

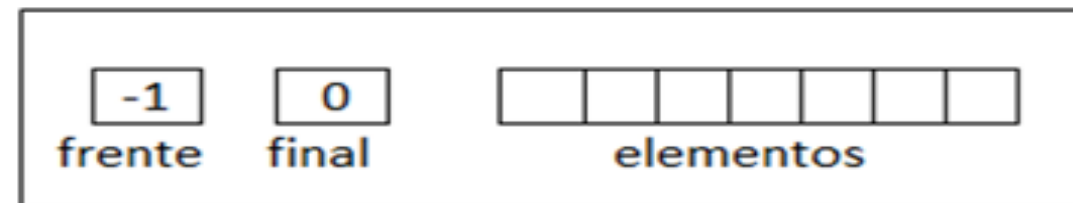
1. La cola de clientes en el banco, en terminal de autobuses, líneas aéreas, etc.
2. Cola para cargar gasolina, para pagar casetas de cobro.
3. Cola de alumnos en la biblioteca, librería, etc.
4. Cola en un supermercado, etc.

2. Implementación estática

Representación de una cola estática usando arreglos.



```
#define TAM 5  
typedef struct{  
    int frente, final;  
    int elementos[TAM];  
}tipoCola;
```



cola

2. Implementación estática

Realizar una prueba de escritorio

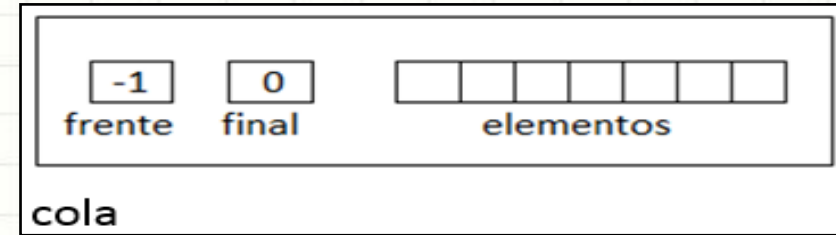


```
#define TAM 5
```

```
typedef struct{  
    int frente, final;  
    int elementos[TAM];  
}tipoCola;
```

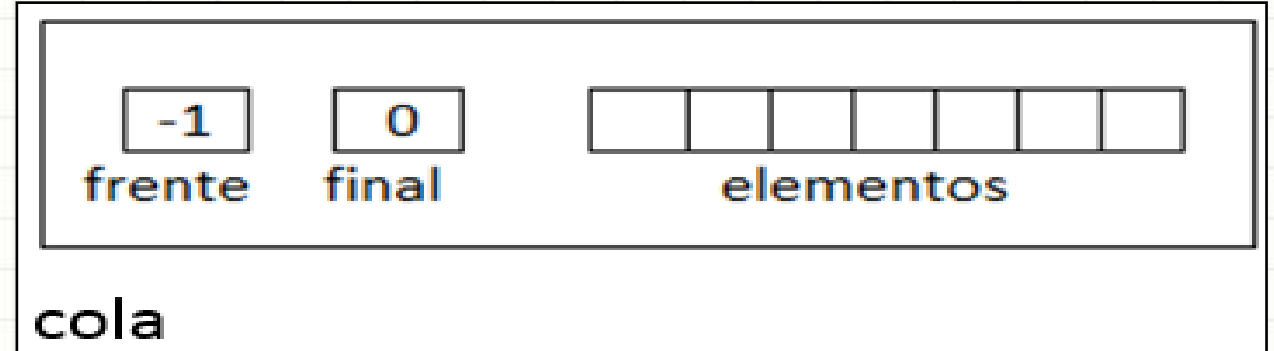
```
void insertar(tipoCola *colaT, int dato);  
int eliminar(tipoCola *colaT);
```

```
int main(){  
    int dato;  
    tipoCola cola={-1,0,{0}};  
    insertar(&cola, 1); //considerar validaciones  
    insertar(&cola, 3);  
    insertar(&cola, 6);  
    dato = eliminar(&cola);  
    ...
```



2. Implementación estática de una cola en línea recta

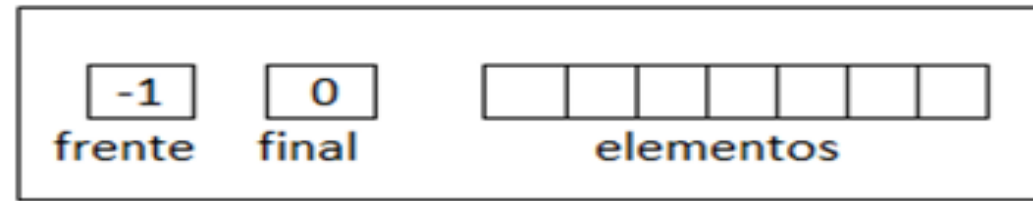
Realizar una prueba de escritorio



```
void insertar(tipoCola *colaT, int dato){  
    //validar invocando a función llena  
    colaT->elementos[colaT->final]=dato;  
    colaT->final++;  
}
```


2. Implementación estática de una cola en línea recta

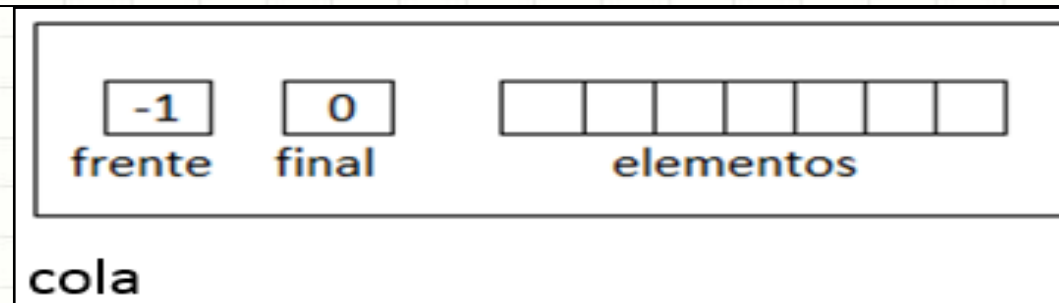
Realizar una prueba de escritorio



cola

```
int eliminar(tipoCola *colaT){  
    int dato;  
    //validar invocando a funcion vacía  
    colaT->frente++;  
    dato=colaT->elementos[colaT->frente];  
    return dato;  
}
```

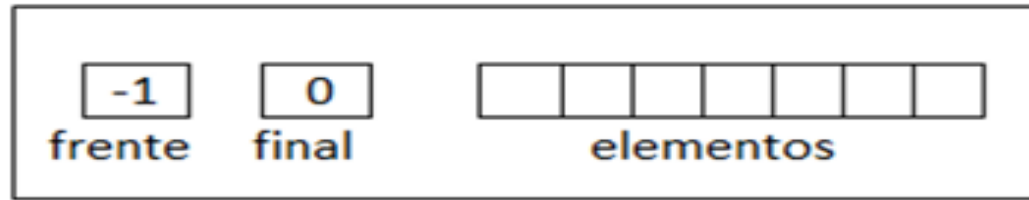
2. Implementación estática de una cola en línea recta



```
//función que determina si la cola está llena  
int llena(tipoCola colaT){  
    int bandera=0;  
    if (colaT.final==TAM)  
        bandera= 1;  
    return bandera;  
}
```



2. Implementación estática



cola

```
int vacia(tipoCola colaT){  
    int bandera=0;  
    if ((colaT.frente)+1 == colaT.final)  
        bandera= 1;  
    return bandera;  
}
```

Realice una prueba de escritorio.



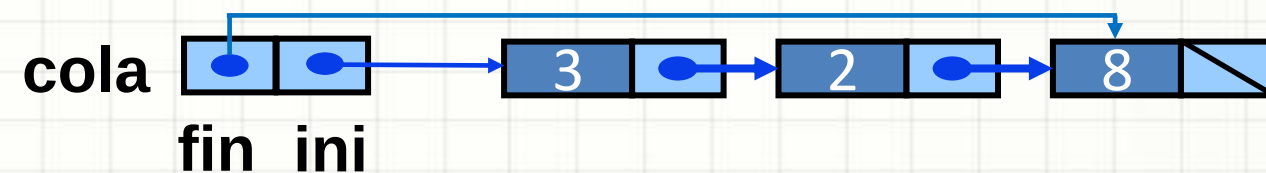
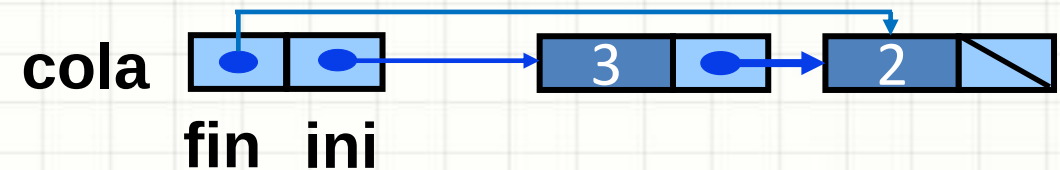
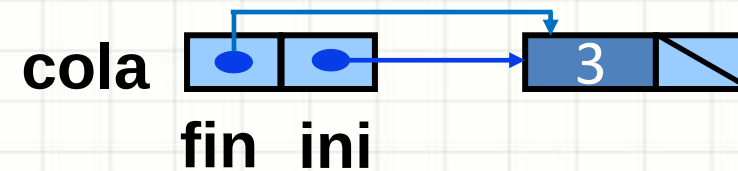
3. Implementación dinámica: tipos de datos

```
typedef struct nodo{
    int dato;
    struct nodo *sig;
}nodoCola;

typedef struct{
    nodoCola * ini;
    nodoCola * fin;
}tipoCola;

typedef tipoCola * tipoColaPtr;
typedef nodoCola * nodoColaPtr;
```

Insertar a la cola vacía, el 3, 2 y 8.



3. Implementación dinámica: tipos de datos y main



```
typedef struct nodo{
    int dato;
    struct nodo *sig;
} nodoCola;

typedef struct{
    nodoCola * ini;
    nodoCola * fin;
}tipoCola;

typedef tipoCola * tipoColaPtr;
typedef nodoCola * nodoColaPtr;

void insertar(tipoColaPtr colaT, int dato);
int eliminar(tipoColaPtr colaT);
```

```
int main(){
    tipoCola cola={NULL};
    int dato;

    insertar(&cola, 3);
    insertar(&cola, 2);
    insertar(&cola, 8);
    dato=eliminar(&cola);
    ...
}
...
```

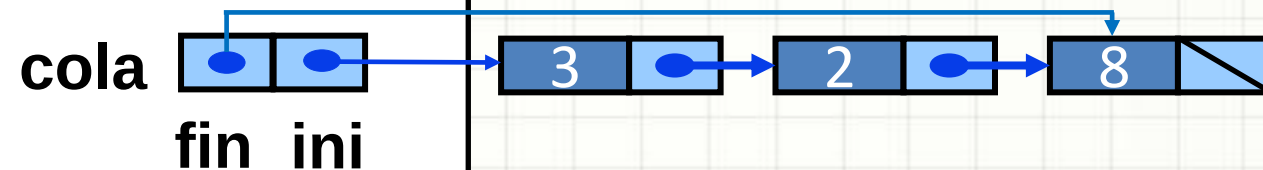
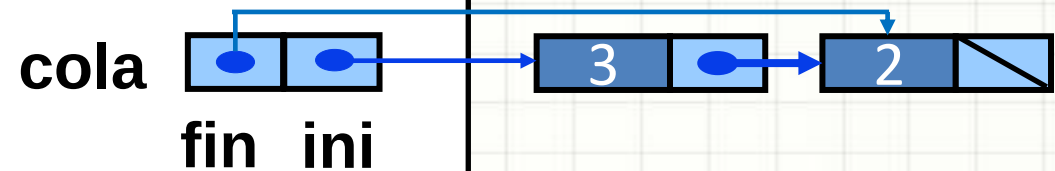
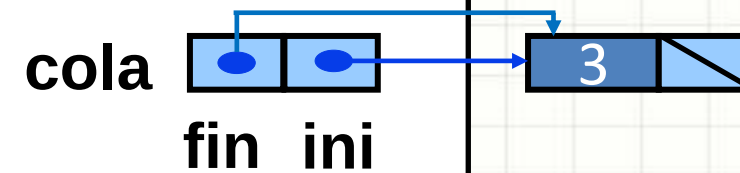


```

void insertar(tipoColaPtr colaT, int dato){
nodoColaPtr nuevo;
nuevo=(nodoColaPtr)malloc(sizeof(nodoCola));
if(nuevo==NULL)
    printf("no hay memoria\n");
else {
    nuevo->dato=dato;
    nuevo->sig=NULL;
    if(colaT->ini==NULL)
        colaT->ini=nuevo;
    else
        colaT->fin->sig=nuevo;
    colaT->fin=nuevo;
}}

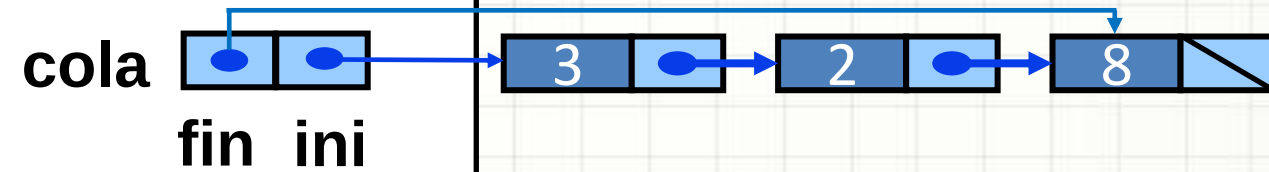
```

3. Implementación dinámica: insertar



```
//previamente validar que la cola no  
//esté vacía para invocar a esta función.
```

```
int eliminar(tipoColaPtr colaT) {  
    nodoColaPtr temp;  
    int dato;  
    temp=colaT->ini;  
    dato=temp->dato;  
    if(colaT->ini==colaT->fin)  
        colaT->fin=temp->sig;  
    colaT->ini=temp->sig;  
    free(temp);  
    return dato;  
}
```



3. Implementación dinámica: eliminar



4. Colas de prioridad

Tanto la **pila** como la **cola** son estructuras de datos cuyos elementos están ordenados conforme se insertaron.

La **cola de prioridad** es una estructura de datos en la que el ordenamiento intrínseco de los elementos determina los resultados de sus operaciones básicas

Tipos de colas de prioridad:

- Cola de prioridad ascendente
- Cola de prioridad descendente



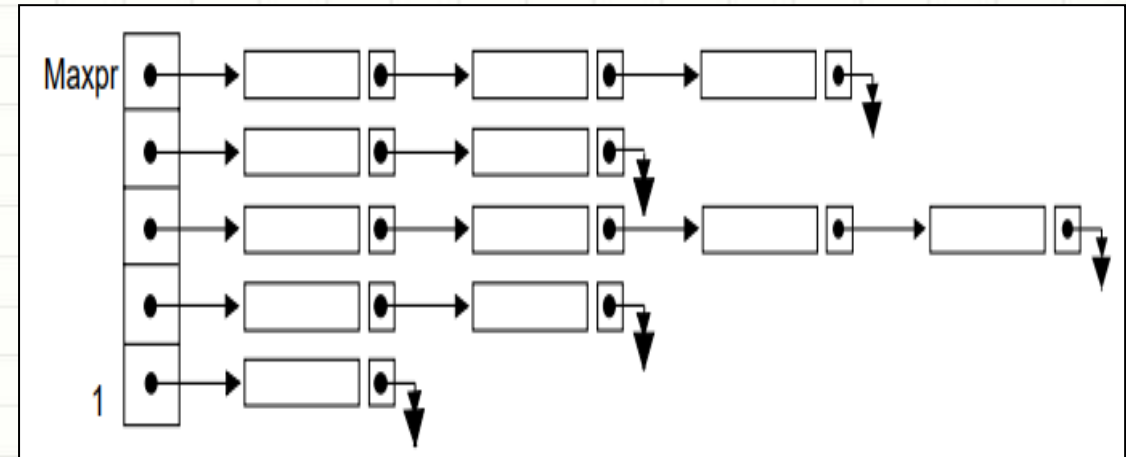
4. Colas de prioridad

Cola de prioridad ascendente:

Colección de elementos en la que pueden insertarse elementos de manera arbitraria y de la que puede eliminarse sólo el elemento menor.

Cola de prioridad descendente:

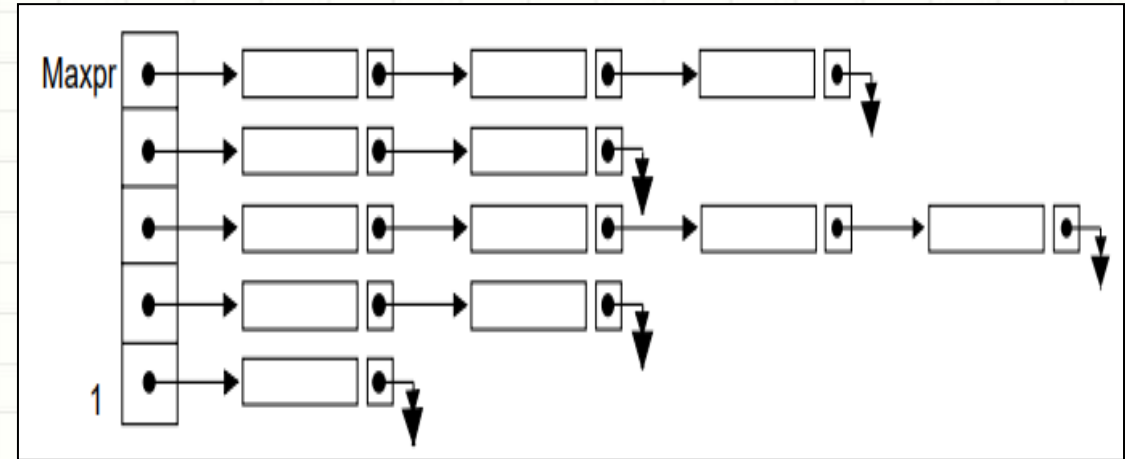
Es similar a la de prioridad ascendente, pero sólo permite la eliminación del elemento mayor.



4. Colas de prioridad: Implementación dinámica de documentos a imprimir

Ejemplo, insertar los siguientes documentos:

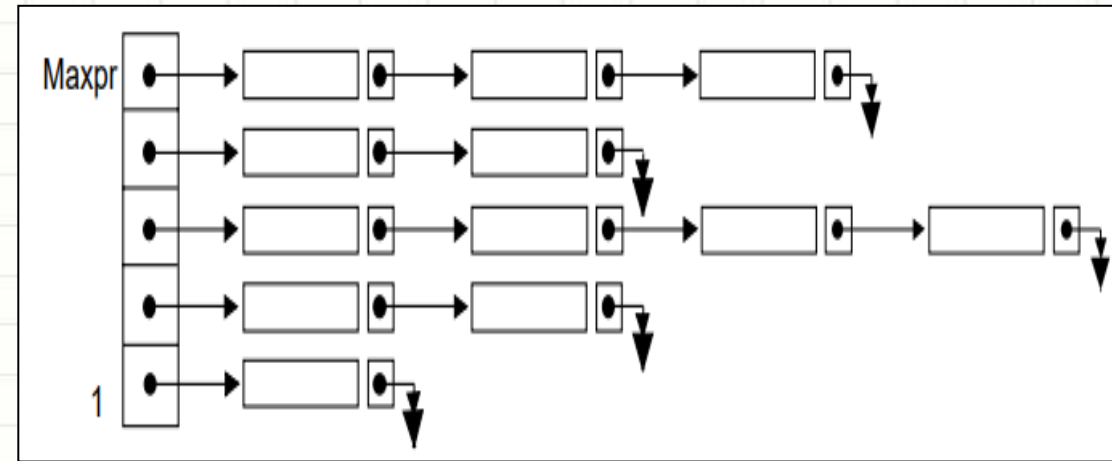
{docto1, 1}, {docto2, 2}, {docto3, 1},
{docto4, 3}, {docto5, 2}, etc.



```
typedef struct documento{  
    char nombre[20];  
    int prioridad; //ej. 1-3  
    struct documento * sig;  
}nodoD;  
typedef struct{  
    nodoD *ini;  
    nodoD *fin;  
}tipoCola;
```

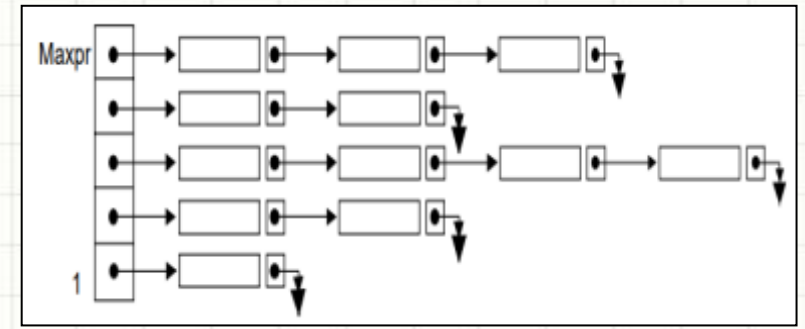

4. Colas de prioridad: Implementación dinámica de documentos a imprimir

```
typedef struct documento{  
    char nombre[20];  
    int prioridad; //ej. 1-3  
    struct documento * sig;  
} nodoD;  
  
typedef struct{  
    nodoD *ini;  
    nodoD *fin;  
}tipoCola;
```



```
int main(){  
    tipoCola colasPrioridad[3]={NULL};  
    ...
```

4. Colas de prioridad: Implementación dinámica de documentos a imprimir



Considere la definición anterior de los tipos de datos para colas de prioridad, escriba un programa que contenga el siguiente menú de opciones:

1. Insertar un elemento en la cola de prioridad correspondiente.
2. Imprimir todos los elementos que están al inicio de cada cola de prioridad.
3. Imprimir todos los elementos que están al final de cada cola de prioridad.
4. Eliminar un elemento de la cola con menor prioridad.
5. Salir (en caso que haya elementos en la colas, los elimine y despliegue cada uno de ellos).

Casos de estudio

- Sistema bancario (ventanilla)
- Sistema de un supermercado (cola de cajas)
- Simulación de un cine

Simulación de venta de boletos de un cine (Continuación)

Implementar un sistema que calcule los ingresos de la venta de boletos de un cine mediante **colas dinámicas de prioridad**.

Los datos de las películas se deben de leer de un archivo y guardar en un arreglo de estructuras de tipo de dato **tipoPelicula** que contiene los siguientes campos: idPelicula, nombre, costoEntrada, numSala(prioridad) y fecha (leída del sistema).

La cola de clientes se debe formar con nodos(tipo de dato **tipoNodo**) compuestos de los siguientes campos: pelicula(tipoPelicula) y apuntador siguiente(tipoNodo).

Simulación de venta de boletos de un cine (Continuación)

Las funcionalidades que se implementarán son las siguientes:

- **1. Simular entradas** (función con paso por referencia): De manera aleatoria se generarán e insertaran a las colas de prioridad, n entradas.
- **2. Mostrar las entradas** (función con paso por valor): listar por pantallas todas las entradas por número de sala, para esta función puede recorrer las colas sin aplicar exactamente insertar y eliminar de una cola
- **3. Simular la venta de todas las entradas** (función con paso por referencia): Se van sacando las entradas de la cola, imprimiendo los datos correspondientes y se generan los ingresos por número de sala, fecha y finalmente se imprimen el total de ingresos por sala y el total de ingresos de todas las salas.
- **4. Consulte que película es la más taquillera.**

Referencias

1. Langsam Y., Augenstein M. J., Tenenbaum A. M. Estructuras de Datos en C. Prentice-Hall, México 1997.
2. Wirth, Niklaus. Algoritmos y estructura de Datos. Prentice-Hall, México.
3. Joyanes A. L, et. al. Estructuras de datos en C. McGraw-Hill, 2005.
4. Cairó O., Guardati S. Estructuras de datos . 3ª edición. McGraw-Hill, 2010.