TAD Pilas y Colas sobre vectores

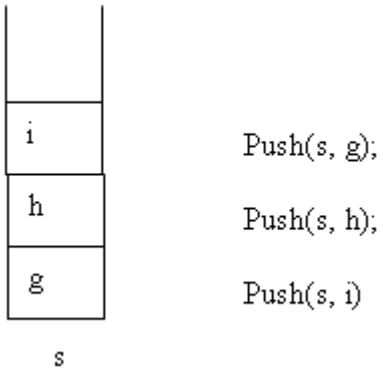
Pilas

Una pila es un tipo especial de estructura de datos en la que sólo se pueden insertar y eliminar elementos en uno de los extremos de la estructura. Estas operaciones se conocen como "push" y "pop", respectivamente "apilar" y "desapilar". Además, las escrituras de datos siempre son inserciones de elementos, y las lecturas siempre eliminan el elemento leído.
Estas características implican un comportamiento de lista LIFO (Last In First Out), el último en entrar es el primero en salir.

El símil del que deriva el nombre de la estructura es una pila de platos. Sólo es posible añadir platos en la parte superior de la pila, y sólo pueden tomarse del mismo extremo.

Operaciones con Pilas

Push: Se utiliza para introducir elementos, ésta función necesita como argumentos de entrada a la pila y el elemento a introducir. La función retornará "verdadero" (1) si la opración de apilar pudo hacerse con éxito, o "falso" (0) si no pudo apilarse.
Pop: se utiliza para sacar elementos de la pila, como argumento de entrada, necesita, únicamente a la pila, y como argumento de salida (pasado por referencia) devolverá el elemento que extrae de la pila. La función retornará "verdadero" (1) si la opración de desapilar pudo hacerse con éxito, o "falso" (0) si no pudo desapilarse.



Hemos llenado nuestra pila, llamada "s", con los valores que le indicamos en la función push.
Ahora si queremos sacar el elemento del tope, basta indicar con la siguiente sentencia:
Pop(s,d);
Pop(s,d);
Y la pila quedaría:

Se debe tener en cuenta que, dentro de las funciones pop y push, pueden estar otras funciones inmersas, por ejemplo, si queremos introducir más datos, debemos tener otra función que verifique si la pila, no está llena. Por el contrario si queremos sacar datos, debemos cerciorarnos

que la pila no esté vacía, es ahí donde nacen estas otras funciones que deben estar presentes en nuestros programas.

Algoritmo de la función push (para arreglos)

1. verificar si la pila no está llena
2. incrementar en 1 el puntero índice de la pila
3. almacenar el elemento en la posición del puntero de la pila

Algoritomo de la función pop (para arreglos)

1. si la pila está vacía imprimir un mensaje
2. sino está vacía, leer el elemento de la posición del puntero de la pila
3. decrementar en uno el puntero de la pila.

Las pilas se pueden implementar de dos formas:
- Como Arreglos
- Usando Memoria Dinámica.

```
# include <iostream>
using namespace std;
#define N 30
typedef struct pila
{
   int datos[N];
   int tope;
};
void inicializar(pila &p)
{
   p.tope=0;
}

int pilallena(pila p)
{
   if (p.tope==N)
      return 1;
   else
      return 0;
}

int pilavacia(pila p)
{
   if (p.tope==0)
      return 1;
   else
      return 0;
}

int apilar(pila &p, int dato) // push
{
   if (pilallena(p)!=1)
```

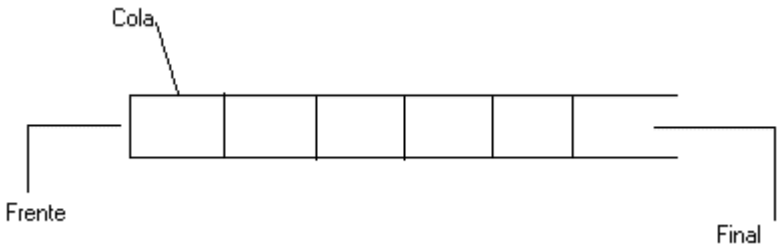
```
   {
      p.datos[p.tope]=dato;
      p.tope++;
      return 1;
   }
   else
      return 0;//fallo la operacion.
}

int desapilar(pila &p,int &dato) //pop
{
   if (pilavacia(p)!=1)
   {
      dato=p.datos[p.tope-1];
      p.tope--;
      return 1;
   }
   else
      return 0;
}

int main()
{
   pila mipila;
   int valor;
   inicializar(mipila);
   apilar(mipila,10);
   apilar(mipila,20);
   apilar(mipila,30);
   desapilar(mipila,valor);
   printf("%d\n",valor);
   apilar(mipila,40);
   desapilar(mipila,valor);
   desapilar(mipila,valor);
   printf("%d\n",valor);
   system("PAUSE");
   return 0;
}
```

Colas

Una estructura muy utilizada e importante en el área de la programación es la estructura de datos tipo Cola.
Por ejemplo cuando mandamos a impresión un documento, éste actúa en forma de cola, es decir; primera en entrar, primera en salir, o por sus siglas en inglés FIFO (First in, first out), ya que el primer documento que llega a la "Cola de Impresión", es el primer documento que es impreso.
Pero las colas, no sólo son usadas por las impresoras, sino también en la vida cotidiana tienen otras muchas aplicaciones, por ejemplo el celular, cuando recibimos un mensaje de texto, éste es almacenado en un "lugar" (dentro del chip) que se comporta como cola; otro ejemplo son las colas de tareas en la pc, las colas de prioridades, etc.
Claro, estos ejemplos son muy complejos y exigen que tengamos conocimientos de sistemas operativos (uso y creación), por tanto no vamos a entrar en tanto detalle respecto a las aplicaciones complejas de las colas, sin embargo trataremos algunas abstracciones que nos ayudarán a comprender el funcionamiento de ésta estructura.

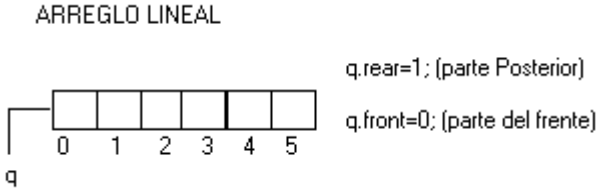
Concepto de Cola

Una cola, es una estructura en la cual se almacenan elementos (en orden de llegada), es decir que, se ingresan los elementos por la parte final de la estructura y se eliminan (o se sirven) por la parte del frente.



Como puede observarse, ésta estructura cuenta con dos apuntadores, uno que apunta al último elemento y otro que apunta hacia el primer elemento (o el elemento del frente).
Se debe tener en cuenta que, una cola, almacena los datos en ella, manteniendo cierto orden, ya que sus elementos se añaden por el final de la cola y se extraen o se eliminan por la parte de frente.
El lector prensará el por que la insistencia de ello, pero cuando uno inicia este estudio, al momento de programar, se confunde fácilmente, por que las sentencias de las funciones son muy parecidas a la de una pila.

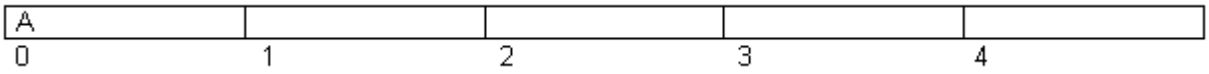
Tipo Cola Implementado como Arreglo



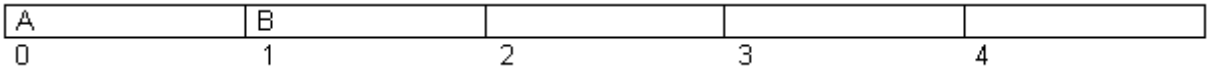
La figura de arriba, muestra la forma de implementar una cola, como arreglo, en la que cada casilla, representa una estructura compuesta por el tipo de dato a guardar (o bien otra estructura).

Las variables q.rear y q.front, se van modificando cada vez que añadimos o eliminamos datos de nuestra cola.

Ahora vemos un ejemplo del funcionamiento de las colas, implementadas como arreglos:
Supongamos que en una cola vamos a almacenar elementos de tipo carácter.
Insert(&q, 'A');

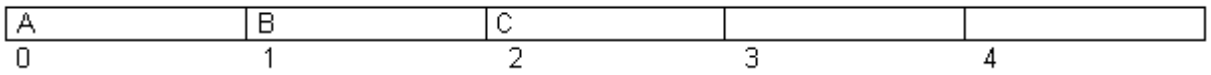


q.rear=0, q.front=0
insert (&q, 'B')



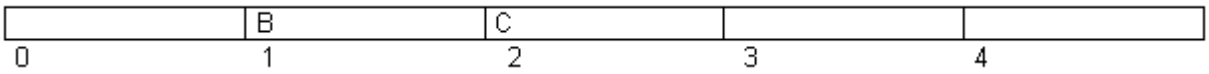
q.rear=1
q.front=0

insert (&q, 'C');



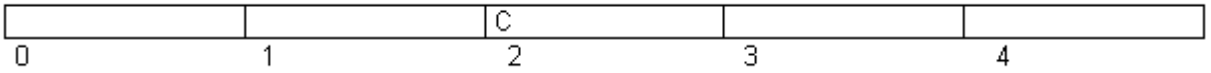
q.rear=2
q.front=0

remove(&q);



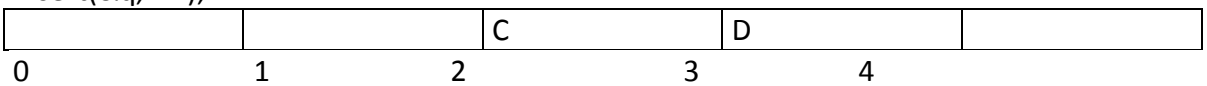
q.rear=2
q.front=1

remove(&q);



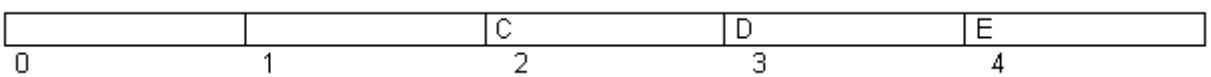
q.rear=2
q.front=2

insert(&q, 'D');



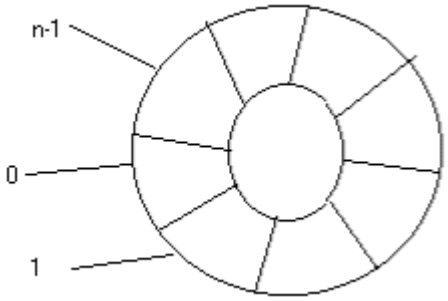
q.rear=3
q.front=2

insert(&q, 'E');



q.rear=4
q.front=2
Como se puede ver, al manejar una cola en forma de arreglo lineal, resulta complicado y poco eficiente, por que al eliminar elementos quedan nodos vacíos y cuando se llega al último elemento de la cola, aparecerá un mensaje de error, (o al menos debería aparecer), que indique que ya no

hay más espacios, sin embargo, eso sería una total mentira, ya que tenemos elementos sin utilizar; una solución para ello podría ser que cada vez que se eliminen un elemento mover todos los datos hacia la izquierda, pero eso es complejo (y no resuelve el problema de la eficiencia), entonces es donde surge el considerar la cola, como un arreglo circular, de modo que el último elemento del arreglo se conecta con el primer elemento del arreglo, mediante la lógica del algoritmo.




```
# include <iostream>
using namespace std;

const int N=3;
struct cola
{
        int ini; //rear
        int ult; // front
        int cant; // cantidad de elementos actualmente cargados en la cola
        int valores[N];
} ;

void inicializar(cola &c)
{
  c.ini = 0;
  c.ult = 0;
  c.cant = 0;
}

int colallena(cola c)
{
        if(c.cant == N)
                return 1;
        else
                return 0;
}

int colavacia (cola c)
{
        if (c.cant==0)
                return 1;
        else
                return 0;
}
```

```
int agregar (cola &c, int elem)
{
	if (colallena(c)!=1) // si la cola no esta llena
	{	c.valores[c.ult]=elem;
		c.cant++;
		if (c.ult != (N-1))
			c.ult++;
		else
			c.ult=0;
		return 1;
	}
	else
		return 0;
}

int quitar (cola &c, int &elem)
{
	if (colavacia(c)!= 1) // cola no esta vacia
	{	elem = c.valores[c.ini];
		c.cant--;
		if (c.ini != (N-1))
			c.ini++;
		else
			c.ini=0;
		return 1;
	}
	else
		return 0;
}

int main()
{
	int e;
	cola cola1;
	inicializar(cola1);
	agregar(cola1, 10);
	agregar(cola1, 20);
	agregar(cola1, 30);
	agregar(cola1, 40);
	quitar(cola1, e);
	printf("%d \n",e );
	agregar(cola1, 50);
	quitar(cola1, e);
	printf("%d \n",e );
	quitar(cola1, e);
	printf("%d \n",e );
	quitar(cola1, e);
	printf("%d \n",e );
```

```
        system("PAUSE");
    return 0;
}
```