Cuarta Unidad

Pilas y Colas

Sumario

Comprende y sintetiza los conceptos asociados a las pilas y colas. Se abstrae, identifica y clasifica las características y acciones básicas (primitivas) relacionadas a las pilas y colas.

- Pilas y colas

Lección 7

Pilas y Colas

El tener una estructura dinámica significa tener un conjunto de elementos relacionados de tal manera que en tiempo de ejecución de un programa, la estructura pueda crecer o reducirse. En una pila y en una cola se espera este tipo de comportamiento ya que ambas son consideradas estructuras de un carácter dinámico.

7.1. Pila

La pila se utiliza siempre que se quiera recuperar una serie de elementos en orden inverso a como fueron introducidos. La eliminación de un elemento de una pila se realiza por la parte superior, lo mismo que la inserción, la pila también se conoce con el nombre de LIFO (Last Input, First Output: último en entrar, primero en salir). "A diferencia del arreglo la definición de pila incorpora la inserción y supresión de elementos, de tal manera que esta es un objeto dinamico constantemente variable" (Tenenbaum, 1993, p. 68).

7.1.1. Operaciones con pilas

Dos operaciones son fundamentales en una pila:

- Meter(push) que adiciona un elemento a la pila.
- Sacar(pop) que cumple la operación inversa de push, osea retira un elemento de la pila.

Otras operaciones usualmente incluidas en una pila son:

- Verificar si esta ocupada o llena la pila
- Verificar si la pila se encuentra vacia

7.1.2. Aplicaciones de pilas

- Navegador Web.
  - Su utilidad se hace presente cuando usando el concepto de pila, los sitios que son visitados en el proceso de navegar se apilan uno sobre otro y cuando se desea retornar a un sitio visitado anteriormente se extrae de la pila al presionar o elegir la opción retroceso.

- Editores de texto
  - Los cambios efectuados se almacenan en una pila.

- Los usuarios pueden deshacer los cambios mediante una operación deshacer o "undo" en inglés, la cual extrae el estado del texto antes del ultimo cambio realizado.

"Cuando se tiene un programa que llama a un subprograma, internamente se usan pilas para guardar el estado de las variables del programa en el momento que se hace la llamada. Asi, cuando termina la ejecución del subprograma, los valores almacenados en la pila pueden recuperarse para continuar con la ejecución del programa en el punto en el cual fue interrumpido. Ademas de las variables debe guardarse la dirección del programa en la que se hizo la llamada, porque es a esa posicion a la que regresa el control del proceso" (Cairó, 2002, 86)

7.1.3. Implementación de una pila

Las pilas se pueden implementar mediante:

- Arreglos
- Listas enlazadas
- Filas secuenciales

7.1.3.1. Implementación mediante listas

Una implementación de una pila usando listas no presenta las desventajas de una pila implementada bajo arreglos sin embargo presenta otros inconvenientes que veremos a continuación:

Ventajas

- Una pila basada en listas y por el carácter dinámico que poseen las listas puede hacer posible una pila que en tiempo de ejecución varie su tamaño o tal caso diseñar una pila sin limites de tamaño.
- Una pila bajo una implementación de listas dinamicas solo ocupara el espacio que necesita, a diferencia de un arreglo en el cual puede existir desperdicio de espacio.

Desventajas

- El acceso es secuencial en una lista, razón por la cual aunque se tenga un apuntador en la cima de la pila para gestionar de manera mas rápida la entra o salida de elementos, siempre será mas lento que la manera en que lo hace un arreglo.
- Hay tiempo que se consume en el proceso de gestionar memoria para cada nodo que se ingresa, asi como también en el proceso de destruir un elemento cuando se efectua una operación sacar.
- Se usa información adicional para las referencias o enlaces.

Especificación de los TAD's y los algoritmos

Especificación PILA
variable
numero : ENTERO.
raiz : PILA.
sgte : PILA.
tope : entero.
i : entero.
métodos
PILA(tamaño) : no retorna valor.
pilaLlena() : retorna valor lógico.
pilaVacia() : retorna valor lógico.
meterPila(dato) : no retorna valor.
sacarPila() : retorna valor entero.
mostrarPila() no retorna valor,
significado
pilaLlena retorna verdadero si la pila llego al tope de su tamaño, en caso contrario retorna falso.
pilaVacia retorna verdadero si la pila esta vacia, en caso contrario retorna falso.
meterPila ingresa un elemento a la pila.
sacarPila saca un elemento de la pila.
mostrarPila muestra todos los elementos de la pila.
Fin_especificación

```
Procedimiento meterPila( dato)
        PILA :ptr
        crear(ptr)
        Asignar(ptr,dato)
        Sgte(ptr)← raiz
        raiz ← ptr
        i ←i+1
Fin_procedimiento

Funcion sacarPila(): entero
        PILA :ptr
        entero cima
        ptr←sgte(raiz)
        leer(raíz, numero)
        cima←numero
        liberar( raiz )
        raiz←ptr
        i←i-1
        retornar cima
Fin_funcion
```

Implementación del TAD

```
#include <iostream>
```

```
#include <iomanip>
#include <cstdlib>
using namespace std;

class PILA
{
	int numero;
	PILA *sig,*raiz;
	int tope;
	int i;
	public:
	PILA(){ }
	PILA(int TAMANO)
	{
		tope=TAMANO;
		raiz=NULL;
		i=0;
	}
	void menu(){
		cout<< "\nMENU DE OPCIONES\n";
		cout<< "----------------\n" ;
		cout<<"<1> Meter	\n";
		cout<<"<2> Sacar	\n";
		cout<<"<3> Mostrar	\n";
		cout<<"<4> Salir	\n";
	}
	bool pilaLLena()
	{
		if (i>=tope)
			return true;
		else
			return false;
	}
	bool pilaVacia()
	{
		if (i<=0)
			return true;
		else
			return false;
	}
	void meterPila(int dato)
	{
		PILA *ptr=new  PILA;
		ptr->numero = dato;
		ptr->sig  = raiz;
		raiz = ptr;
		i++;
	}
```

```
        int sacarPila()
        {
                PILA *ptr; int cima;
                ptr=raiz->sig;
                cima=raiz->numero;
                delete raiz;
                raiz=ptr;
                i--;
                return cima;
        }
        void mostrarPila()
        {
                PILA *ptr;
                ptr=raiz;
                while( ptr!= NULL)
                {
                        cout<<"   "<<ptr->numero;
                        ptr=ptr->sig;
                }
                cout<<endl;
        }
};
int main()
{
        char opcion;
        int dato;
        PILA pil(4);
        do
        {
                pil.menu();
                cout<<"\n Ingrese opcion : ";
                opcion=cin.get();
                switch(opcion)
                {
                        case '1':
                                cout<<"ingreso un numero entero a la pila:
                        ";cin>>dato;
                                if (!pil.pilaLLena())
                                        pil.meterPila(dato);
                                else
                                        cout<<"pila llena";
                                break;
                        case '2':
                                if (!pil.pilaVacia())
                                        cout<<"\n sale= "<<pil.sacarPila();
                                else
                                        cout<<"pila vacia ";
                                break;
```

```
                case '3':
                        pil.mostrarPila();
                        break;
            }
            cin.ignore();
        }
        while( opcion !='4');
        system("PAUSE");
        return 0;
}
```

7.2. Colas

En una cola a diferencia de una pila la operación de ingreso o encolar se hace por un extremo, según se observa en la grafica se hace por el lado izquierdo o final de la cola y la operación retirar o desencolar se hace por el otro extremo denominado frente o principio de la cola. El primer elemento que entra en la cola será el primero en salir, debido a esta característica la cola también recibe el nombre de estructuras FIFO (first-in, first-out, primero en entrar, primero en salir)

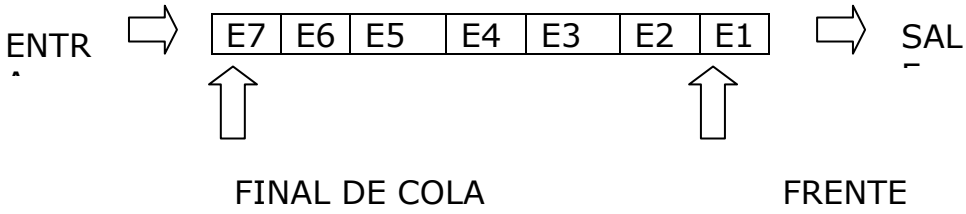


Figura 7.1. Estructura de una cola

"En ingles ordinario una cola se define como una línea de espera, como una fila de personas en espera de adquirir boletos, donde la primera persona en la fila es la primera en ser atendida. Para aplicaciones de computadora, definimos de modo similar la cola como una lista en la que todas las adiciones a la lista se hacen en un extremo, y todas las eliminaciones se efectúan en el otro extremo" (Kruse, 1988, 47)

7.2.1. Operaciones con colas

Una cola es una estructura a la que usualmente se le da un tamaño. Esta estructura puede estar en un momento vacia, estar llena hasta el tope o a medio llenar. Se hace necesario por ello para manipular este TAD de cuatro operaciones.

Las operaciones básicas de un TAD cola son :

- Ingresar o encolar es la operación que añade un nuevo elemento a la cola.
- Sacar o desencolar es la operación que elimina el elemento del frente de la cola

Otras operaciones usualmente incluidas en el tipo de dato abstracto cola son:

- Verificar si la cola esta llena
- Verificar si la cola esta vacia.

7.2.2. Aplicaciones de colas

- "Otro ejemplo se refiere a las redes de computadores. Hay muchas redes de computadores personales en las que el disco esta conectado a una maquina, conocida como servidor de archivos. Los usuarios en otras maquinas obtienen acceso a los archivos sobre la base de que el primero en llegar es el primero atendido, asi que la estructura de datos es una cola." (Allen, 1995, 87)
- Clientes solicitando ser atendidos por una telefonista.
- Simulaciones de cualquier situación real en la que se presenta una organización tipo cola.
- En la ciencia de la Investigación Operativa es muy usado el concepto de colas ya que como modelo matemático que es la teoría de colas, y una cola es lo que se conoce también como línea de espera, el tema líneas de espera no es más que el uso de las colas para explicar sistemas completos de líneas de espera individuales que tienen sus propias características.

7.2.3. Implementación de una cola

Las colas se pueden implementar mediante:

- Arreglos.
- Listas enlazadas.

7.2.3.1. Implementación de colas mediante listas

Cuando usamos un TAD lista enlazada dinámicamente es posible que pueda operar muy bien en ciertos casos pero ocacionar otros tipos de problemas. Si trabajamos una cola usando una lista enlazada dinámica en la cual los elementos serán ingresados o retirados varias veces o muy frecuentemente y es importante el lugar de los nuevos elementos que se ingresan entonces el uso de una lista de este tipo para la implementación de la cola será muy conveniente.

Ventajas

- Al tener una cola implementada bajo una lista enlazada dinámicamente y teniendo la capacidad de ingresar elementos de manera indefinida, posibilita que el tamaño de la cola pueda variar en tiempo de ejecución, reduciéndose o aumentando, a diferencia de lo que no sucede en una cola implementada bajo un arreglo.
- En una cola bajo este tipo de implementación a diferencia de una cola bajo un arreglo lineal, el costo de retirar un elemento no obliga a reorganizar los demás elementos de la cola.

Desventajas

- Una cola bajo este tipo de implementación solo permite acceso secuencial. Esto puede traer costo de acceso en el ingreso o salida de un elemento, dependiendo de cómo este estructurada la lista.
- Otra desventaja de una cola bajo este tipo de implementación es que los elementos de la cola se verán obligados a guardar mas información de lo que guardarían en un arreglo. Esta información adicional se refiere a las referencias o apuntadores que son los que permiten el enlace de los elementos con otros elementos.
- Una cola bajo este tipo de implementación donde cada ingreso de un elemento exija el asignar memoria podría resultar en una cola lenta en su ejecución.

Especificación de los TAD's y los algoritmos

Especificacion COLA

variable

numero: entero
raiz : COLA
sgte : COLA
tope :entero
i :entero

métodos

COLA : no retorna valor
COLA(tamaño) : no retorna valor
colaLlena() : retorna valor logico
colaVacia() : retorna valor logico
meterCola(dato) : no retorna valor
sacarCola() : no retorna valor
mostrarCola() : no retorna valor

significado

colaLlena retorna verdadero si la cola llego al tope de su tamaño, en caso contrario retorna falso.
colaVacia retorna verdadero si la cola esta vacía, en caso contrario retorna falso.
meterCola ingreso un elemento en la cola.
sacarCola saca un elemento de la cola.

mostrarCola muestra todos los elementos o datos de la cola.

Fin_especificación

```
Procedimiento meterCola( dato)
        // Definir variables
        puntero: q, ptr
        crearCOLA( ptr)
        asignar(ptr, dato)
        Si ( raiz = nulo) entonces
                raiz ← ptr
        Sino
                q ←raiz
                Mientras (sgte (q)  ≠ nulo) Hacer
                                q ← sgte(q)
                Fin_mientras
                sgte (q) ← ptr
        Fin_si
        sgte(ptr) ←  nulo
        i←i+1
Fin_procedimiento

Funcion sacarCola(): entero
        // Definir variables
        COLA : ptr
        entero :valor
        leer(raíz,valor)
        i←i-1
        ptr←sgte(raíz)
        liberar(raiz)
        raíz←ptr
        retornar valor
Fin_función
```

Implementación del TAD

```
#include <iostream>
#include <iomanip>
#include <cstdlib>
using namespace std;
class COLA
{
        int numero;
        COLA *raiz,*sig;
        int tope;
        int i;
        public:
```

```
COLA(){}
COLA(int tamano)
{
	tope=tamano;
	raiz=NULL;
	i=0;
}
void menu()
{
	cout<< "\nMENU DE OPCIONES\n";
	cout<< "----------------\n" ;
	cout<<"<1> Meter Cola\n";
	cout<<"<2> Sacar Cola\n";
	cout<<"<3> Mostrar Cola\n";
	cout<<"<4> Salir \n";
}
bool colaLLena()
{
	if (i>=tope)
		return true;
	else
		return false;
}
bool colaVacia()
{
	if (i<=0)
		return true;
	else
		return false;
}
void meterCola(int dato)
{
	COLA *q;
	COLA *ptr= new  COLA;
	ptr->numero=dato;
	if ( raiz == NULL )
		raiz = ptr;
	else
	{
		q=raiz;
		while(q->sig!=NULL)
			q=q->sig;
		q->sig=ptr;
	}
	ptr->sig=NULL;
	i++;
}
int sacarCola()
```

```
	{
		COLA *ptr;
		int valor;
		valor =raiz->numero;
		i--;
		ptr=raiz->sig;
		delete raiz;
		raiz=ptr;
		return valor;
	}
	void mostrarCola()
	{
		COLA *ptr;
		ptr=raiz;
		while( ptr!= NULL)
		{
			cout<<"  "<<ptr->numero;
			ptr=ptr->sig;
		}
		cout<<endl;
	}

};
int main()
{
	char opcion;
	int dato;
	COLA p(3);
	do
	{
		p.menu();
		cout<<"\ningrese opcion : ";
		opcion=cin.get();
		switch(opcion){
			case '1':
				cout<<"ingrese un numero entero
			";cin>>dato;
				if (!p.colaLLena())
					p.meterCola(dato);
				else
					cout<<"cola llena ";
				break;
			case '2':
				if(!p.colaVacia())
					cout<<"sale="<<p.sacarCola();
				else
					cout<<"cola vacia";
				break;
```

```
                case '3':
                        p.mostrarCola();
                        break;
        }
        cin.ignore();
    }
    while( opcion !='4');
    system("PAUSE");
    return 0;
}
```

7.3. Ejercicios resueltos

Ejemplo 01

Una cola medieval se comporta como una cola ordinaria, con la única diferencia de que los elementos almacenados en ella se dividen en dos estamentos: nobles y plebeyos. Dentro de cada estamento, los elementos deben ser atendidos en orden de llegada; pero siempre que haya nobles en la cola, estos deben ser atendidos antes que los plebeyos. Se pide escribir un algoritmo usando asignación dinamica o usando arreglos que implemente la cola medieval.

Solución:

```
Funcion buscar(c): logico
    // definir variables
    Puntero: q←raiz
    Mientras ( q ≠ nulo ) hacer
        Si (q->info.comparaPersona(c)) entonces
            retornar true
        Fin_si
        q ← sgte (q)
    Fin_mientras
    retornar false
Fin_funcion
```

Implementación del TAD

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

class PERSONA
{
    char tipo;
    public :
        void ingresarPersona()
```

```
		{
			fflush(stdin);
			cout<<"ingrese un noble (noble (N) o plebeyo(P)):
";
			cin>>tipo;
		}
		void mostrarPersona()
		{
			cout<<" "<<tipo;
		}
		bool comparaPersona(char c)
		{
			return c==tipo;
		}
};
class COLAMEDIOVAL
{
	PERSONA info;
	COLAMEDIOVAL *raiz,*sgte;
	int tope;
	int i;
	public:
		COLAMEDIOVAL(){}
		COLAMEDIOVAL(int tamano)
		{
			tope=tamano;
			raiz=NULL;
			i=0;
		}
		void menu()
		{
			cout<< "\nMENU DE OPCIONES\n";
			cout<< "----------------\n" ;
			cout<<"<1> Meter Cola\n";
			cout<<"<2> Sacar Cola\n";
			cout<<"<3> Mostrar Cola\n";
			cout<<"<4> Salir \n";
		}
		bool colaLLena()
		{
			if (i>=tope)
				return true;
			else
				return false;
		}
		bool colaVacia()
		{
			if (i<=0)
```

```
                return true;
        else
                return false;
}
bool meterCola(PERSONA dato)
{
        COLAMEDIOVAL *q,*ptr= new  COLAMEDIOVAL;
        ptr->info=dato; /* Asgval( ptr, inf  ) */
        if ( raiz == NULL )
                raiz = ptr;
        else
        {
                q=raiz;
                while(q->sgte!=NULL)
                        q=q->sgte;
                q->sgte=ptr;
        }
        ptr->sgte=NULL;
        i++;
}
bool buscarN(char c)
{
        COLAMEDIOVAL *q=raiz;
        while(q!=NULL)
        {
                if(q->info.comparaPersona(c))
                        return true;
                q=q->sgte;
        }
        return false;
}
PERSONA sacarCola()
{
        COLAMEDIOVAL *ptr;
        PERSONA valor;
        valor =raiz->info;
        i--;
        ptr=raiz->sgte;
        delete raiz;
        raiz=ptr;
        return valor;
}
void mostrarCola()
{
        COLAMEDIOVAL *ptr;
        ptr=raiz;
        while( ptr!= NULL)
        {
```

```
                    ptr->info.mostrarPersona();
                    ptr=ptr->sgte;
                }
                cout<<endl;
            }
};
int main()
{
    char opcion;
    PERSONA dato,ente;
    COLAMEDIOVAL p(4),aux(4);
    do
    {
        p.menu();
        cout<<"\ningrese opcion : ";
        opcion=cin.get();
        switch(opcion)
        {
        case '1':
            dato.ingresarPersona();
            if (!p.colaLLena())
                p.meterCola(dato);
            else
                cout<<"cola llena";
            break;
        case '2':
            if (!p.colaVacia()){
                if(p.buscarN('N')){
                    while(!p.colaVacia()){
                        ente=p.sacarCola();
                        if(!ente.comparaPersona('N'))
                            aux.meterCola(ente);
                        else{
                            cout<<"\nsale=";
                            ente.mostrarPersona();
                        }
                    }
                    while(!aux.colaVacia()){
                        p.meterCola(aux.sacarCola());
                    }
                }
                else {
                    cout<<"\n sale=";
                    (p.sacarCola()).mostrarPersona();
                }
            }
            else  cout<<"cola vacia";
            break;
```

```
            case '3':
                p.mostrarCola();
                break;
            }
            cin.ignore();
        }
        while( opcion !='4');
        system("PAUSE");
        return 0;
}
```

Ejemplo 02

Cuando implementamos una cola usando un arreglo lineal encontramos que va a existir un momento en el que el arreglo no podrá contener más elementos, se ha llegado al final . Sin embargo cada vez que sale un elemento de la cola, todos los elementos podrían arrimarse hacia el lado de salida para dejar más espacio, pero el mover demasiados elementos tiende a afectar el rendimiento de la estructura, asi que como alternativa se pide crear un arreglo circular, en el cual se regresa al inicio del arreglo cuando se llega al final del arreglo.

Solución:

- En esta estructura se requiere el uso de dos índices. Uno se denomina frente y otro denominado atrás. Las dos variables al iniciarse el proceso se inicializan a cero.
- El tamaño de la cola esta determinado por tope.
- Para elegir la posición del siguiente elemento de entrada o de salida se determina usando la operación modulo, por ejemplo:

atras ← (atras+1) modulo tope

Con la informacion anterior ejecutamos algunas operaciones de ENCOLAR Y DESENCOLAR.

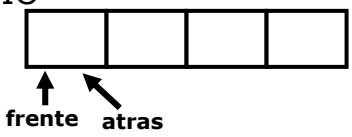
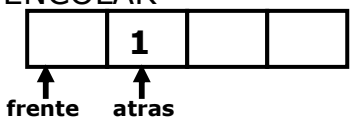
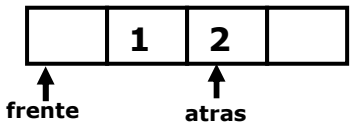
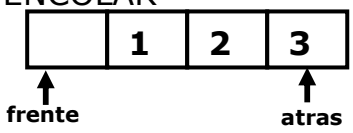
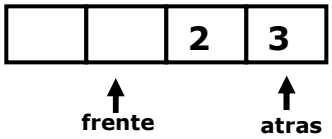
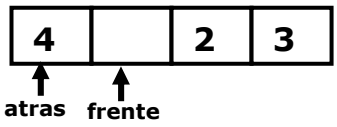
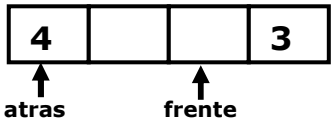
INICIO 	(1) ENCOLAR 
(2) ENCOLAR 	(3) ENCOLAR 
(4) ENCOLAR (no se puede, no hay espacio)	(5) DESENCOLAR 
(6) ENCOLAR 	(7) DESENCOLAR 

Tabla 7.1. Operaciones de encolar y desencolar

Especificación de los TAD's y los algoritmos

Especificacion COLACIRCULAR

variable

v : arreglo de enteros
frente : entero
atras : entero
tope : entero

métodos

COLACIRCULAR(tamaño) : no retorna valor
menu() : no retorna valor
meterCola(x) : no retorna valor
colaVacia() : retorna valor lógico
colaLLena() : retorna valor lógico
sacarCola() : retorna valor entero
mostrarCola() :no retorna valor

significado

COLACIRCULAR inicializa los atributos del TAD.
menu visualiza las operaciones.
meterCola ingresa un elemento a la cola.
colaVacia verifica si la cola esta vacia.
colaLLena verifica si la cola esta llena.
sacarCola saca un elemento de la cola.
mostrarCola muestra los elementos que hay en la cola.

Fin_especificación

```
Procedimiento meterCola(x)
        entero : temp←atras
        atras←(atras+1) modulo tope
        Si(frente=atras) entonces
                atras←temp
        Sino
                v[atras]←x
        Fin_si
Fin_procedimiento

Funcion  sacarCola(): entero
        Si (frente≠atras) entonces
                frente←(frente+1) modulo tope
        Fin_si
        retornar v[frente]
Fin_funcion
```

Implementación del TAD

```
#include <iostream>
#include <iomanip>
#include <cstdlib>
using namespace std;
class COLACIRCULAR{
        int *v;
        int frente,atras;
        int tope;
        public:
        COLACIRCULAR(int tamano){
                v=new int[tamano];
                frente=0;
                atras=0;
                tope=tamano;
        }
        void menu(){
                cout<< "\nMENU DE OPCIONES\n";
                cout<< "----------------\n" ;
                cout<<"<1> Meter        \n";
                cout<<"<2> Sacar        \n";
                cout<<"<3> Mostrar      \n";
                cout<<"<4> Salir        \n";
        }
        void meterCola(int x){
                int temp=atras;
                atras=(atras+1)%tope;
                if(frente==atras){
```

```
                atras=temp;
            }
            else v[atras]=x;
        }
        bool colaVacia(){
            return frente==atras;
        }
        bool colaLLena(){
            return ((atras+1)%tope) == frente;
        }
        int sacarCola(){
            if(frente!=atras)
            frente=(frente+1)%tope;
            return v[frente];
        }
        void mostrarCola(){
            if(frente!=atras)
                if(frente< atras)
                    for(int i=frente; i<atras; i++){
                        cout<<" "<<v[i+1];
                    }
                else {
                    for(int i=frente; i<tope-1; i++)
                        cout<<" "<<v[i+1];
                    for(int i=0; i<=atras; i++)
                        cout<<" "<<v[i];
                }
        }
};
int main(){
        COLACIRCULAR cola(4);
        int valor;
        char opcion;
        do
        {
            cola.menu();
            cout<<"\ningrese opcion : ";
            opcion=cin.get();
            switch(opcion){
            case '1':
                cout<<"\n Ingrese un numero entero a la cola:
            ";cin>>valor;
                if(!cola.colaLLena())
                    cola.meterCola(valor);
                else cout<<"cola ocupada";
                break;
            case '2':
                if(!cola.colaVacia())
```

```
                        cout<<" sale "<<cola.sacarCola();
                    else cout<<"la cola esta vacia:";
                    break;
            case '3':
                    cola.mostrarCola();
                    break;
            }
            cin.ignore();
    }
    while( opcion !='4');
    system("PAUSE");
    return 0;
}
```

Ejemplo 03

Escriba la especificación, el algoritmo y el programa que le permita convertir una expresión en forma infija a una expresión en forma posfija. La expresión de entrada consta de operandos de un solo carácter. Por ejemplo a continuación se muestran algunas expresiones validas:

Expresión infija	Expresión postfija
2+3	23+
(6*2)+7	62*7+
(9*3)+4+6	93*46++

Solución:

Especificación de los TAD's y los algoritmos

Especificación PILA

variable

car	: caracter.
raiz	: PILA.
sgte	: PILA.
tope	: entero.
i	: entero.

métodos

PILA(tamaño)	: no retorna valor.
pilaLlena()	: retorna valor lógico.
pilaVacia()	: retorna valor lógico.
meterPila(dato)	: no retorna valor.
sacarPila()	: retorna valor entero.

significado

pilaLlena retorna verdadero si la pila llego al tope de su tamaño, en caso contrario retorna falso.

pilaVacia retorna verdadero si la pila esta vacia, en caso contrario retorna falso.

meterPila ingresa un elemento a la pila.
sacarPila saca un elemento de la pila.

Fin_especificación

Especificación POSFIJO
variable
pil : PILA.
métodos
POSFIJO(tamaño) : no retorna valor.
operando(s) : retorna valor lógico.
precedencia(o1,o2) : retorna valor lógico.
posfijo(infix, outs) : no retorna valor.
significado
POSFIJO inicializa el tamaño dela pila.
operando verifica si es un operando u operador.
precedencia verifica la precedencia de los operadores.
posfijo cambia la expresión infix infija a una forma posfija en outs.
Fin_especificación

```
Funcion operando(s): logico
    Si (esdigito(s)) entonces
        retornar verdadero  // si es operando
    Sino
        retornar falso // si es operador
    Fin_si
Fin_funcion

Funcion precedencia( o1, o2): logico
    En caso sea (o2) Hacer
        '(': retornar falso
        ')':
            Si (o1='(') entonces
                retornar falso
            Sino
                return verdadero
            Fin_si
    Sino
        retornar falso
    Fin_caso
Fin_funcion

Procedimiento posfijo(infix, outs)
    caracter: sy,symb
    entero : pos←0,on←0
    lógico : SALIO
    sy←infix[pos]
    Mientras (sy≠'\0')  Hacer
```

```
            Si (operando(sy)) entonces
                outs[on]←sy
                on←on+1
            Sino
                Si (no pil.pilaVacia()) entonces
                    symb←pil.sacarPila()
                    SALIO←false
                Sino
                    SALIO←verdadero
                Fin_si
                Mientras((no SALIO)  y (precedencia(symb,sy)))
            Hacer
                    outs[on]←symb
                    on←on+1
                    Si (no pil.pilaVacia()) entonces
                        symb←pil.sacarPila()
                        SALIO←falso
                    Sino
                        SALIO← verdadero
                    Fin_si
                Fin_mientras
                Si (!SALIO) pil.meterPila(symb);
                Si (SALIO || (sy!=')')) pil.meterPila(sy);
                Sino symb=pil.sacarPila();
                Fin_si
                pos←pos+1
                sy←infix[pos]
        Fin_mientras
        Mientras (no pil.pilaVacia()) hacer
            outs[on]←pil.sacarPila()
            on←on+1
        Fin_mientras
        outs[on]←'\0'
Fin_procedimiento
```

Implementación del TAD

```
#include <iostream>
#include <iomanip>
#include <cstdlib>
using namespace std;
class PILA
{
        char car;
        PILA *sig,*raiz;
        int tope;
        int i;
```

```
        public:
        PILA(){ }
        PILA(int TAMANO)
        {
                tope=TAMANO;
                raiz=NULL;
                i=0;
        }
        bool pilaLLena()
        {
                if (i>=tope)
                        return true;
                else return false;
        }
        bool pilaVacia()
        {
                if (i<=0)
                        return true;
                else return false;
        }
        void meterPila(char dato)
        {
                PILA *ptr=new  PILA;
                ptr->car = dato;
                ptr->sig  = raiz;
                raiz = ptr;
                i++;
        }
        char sacarPila()
        {
                PILA *ptr; char cima;
                ptr=raiz->sig;
                cima=raiz->car;
                delete raiz;
                raiz=ptr;
                i--;
                return cima;
        }
};
class POSFIJO{
        PILA pil;
        public:
        POSFIJO( ){
                PILA ini(100);
                pil=ini;
        }
        bool POSFIJO::operando(char s){
                if(isdigit(s)) return true;// si es operando
```

```
        else return false; // si es operador
}
bool POSFIJO::precedencia(char o1,char o2){
        switch(o2){
                case '(':return false;
                case ')':if(o1=='(') return false;
                        else return true;
        default:
                return false;
        }
}
void POSFIJO::posfijo(char infix[],char outs[]){
        char sy,symb;
        int pos=0,on=0;
        sy=infix[pos];
        bool SALIO;
        while (sy!='\0'){
                if (operando(sy)) {
                        outs[on]=sy;
                        on++;
                }
                else{
                        if(!pil.pilaVacia()){
                                symb=pil.sacarPila();
                                SALIO=false;
                        }
                        else SALIO= true;
                        while((!SALIO) && (precedencia(symb,sy))){
                                outs[on]=symb;
                                on++;
                                if(!pil.pilaVacia()){
                                        symb=pil.sacarPila();
                                SALIO=false;
                                }
                                else SALIO= true;
                        }
                        if (!SALIO) pil.meterPila(symb);
                        if (SALIO || (sy!=')')) pil.meterPila(sy);
                        else symb=pil.sacarPila();
                }
                pos++;
                sy=infix[pos];
        }
        while (!pil.pilaVacia()){
                outs[on]=pil.sacarPila();
                on++;
        }
        outs[on]='\0';
```

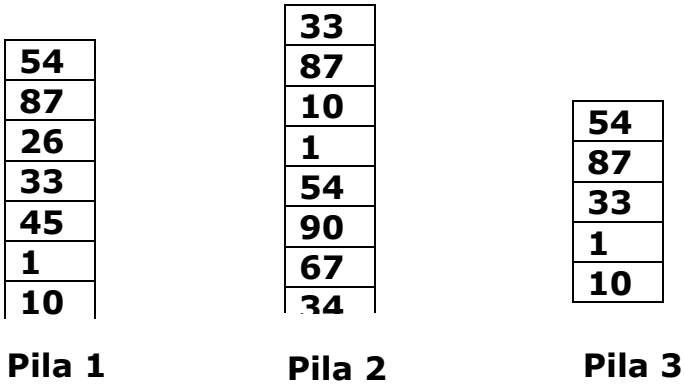
```
        }
};
int main(){
        char cadena[100],salida[100];
        strcpy(cadena,"((3*2)+5)+(1*5)"); // ejemplo de cadena de
entrada
        POSFIJO p;
        p.posfijo(cadena,salida);
        puts(salida);           //devuelve una cadena en posfijo
        system("pause");
        return 0;
}
```

7.4. Ejercicios propuestos

1. Escriba la especificación, el algoritmo y el programa para una pila que verifique el emparejamiento de paréntesis, llaves y corchetes. Por ejemplo una secuencia valida es { { [{ }] } }, en cambio una secuencia incorrecta es { { [} }].

2. Escriba la especificación, el algoritmo y el programa para una cola basada en dos pilas donde las pilas que forman la cola se hagan usando arreglos.

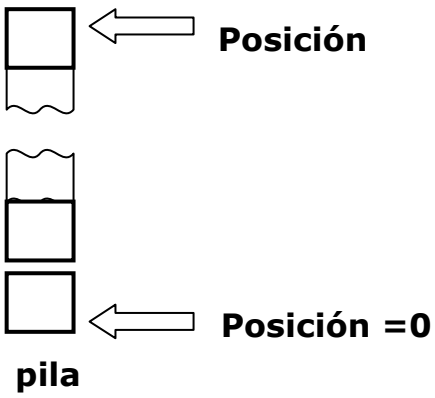
3. Escriba la especificación, el algoritmo y el programa para intersectar dos pilas en una tercera como se observa en el siguiente ejemplo:



4. Escriba la especificación, el algoritmo y el programa para ordenar los valores de una pila. Cuando se ordene la pila se debe respetar la política de ingreso y salida de elementos en la pila.

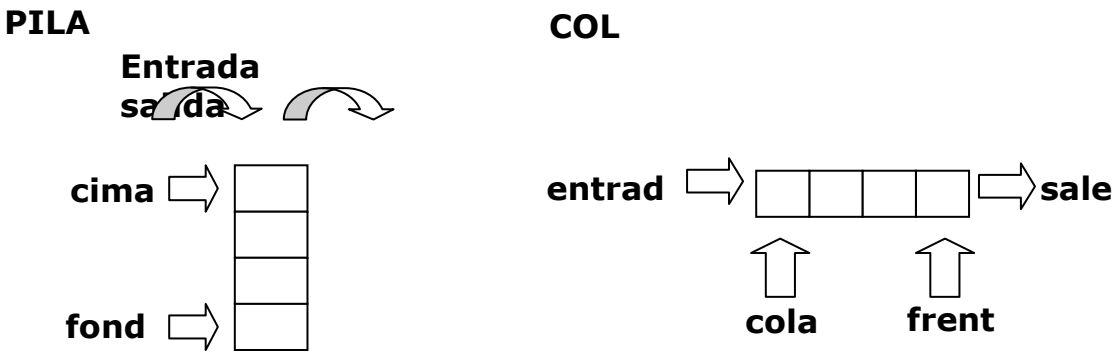
5. Escriba la especificación, el algoritmo y el programa para crear dos pilas usando un mismo arreglo. Una de las pilas se podrá llenar desde la posición 0 hacia el tamañoMaximo-1. La segunda pila comenzara a llenarse desde la posición tamañoMaximo-1 hacia la posicion 0. En la pila tamañoMaximo es la cantidad

máxima de elementos que puede aceptar el arreglo. Controle en la pila que no se sobrepase el tamaño o tope máximo.



Resumen

En esta unidad se tratan dos TAD's que tienen un carácter dinámico y que se diferencian en la forma como se ingresan y retiran los elementos. Una pila es un TAD en el que la eliminación de un elemento de la pila se realiza por la parte superior, lo mismo que la inserción. Se le denomina también LIFO (Last Input, First Output: último en entrar, primero en salir). En una cola la operación de ingreso o encolar se hace por un extremo, y la operación retirar o desencolar se hace por el otro extremo denominado frente o principio de la cola. La cola también se denomina FIFO (first-in, first-out, primero en entrar, primero en salir). Observe abajo gráficamente una pila y un cola:



En ambas estructuras se dan cuatro operaciones básicas: dos operaciones relativas al ingreso y retiro de un elemento y otras dos operaciones para verificar si la estructura esta llena o esta vacia y que se aplican antes de realizar la entrada o salida de un elemento.

Las pilas se aplican en navegadores web, editores de texto, recursividad, etc, a su vez las colas tienen su aplicación en redes de computadoras, simulaciones, clientes solicitando atención en una ventanilla de un banco, etc. Ambas estructuras pueden implementarse mediante técnicas de arreglos o listas enlazadas, presentando cada una de ellas ventajas y desventajas en el tratamiento de la información.

Lectura

Evaluación de expresiones postfijas

Es casi ineludible que la forma prefija llame con tanta naturalidad a una función recursiva para su evaluación, pues la forma prefija es en realidad una formulación "descendente" de la expresión algebraica: las acciones externas y globales se especifican primero y luego en la expresión se hace lo mismo con los componentes. Por otra parte, en la forma postfija los operandos aparecen primero, y después la expresión entera se forma a partir de sus operandos simples y de los operadores internos, en forma "ascendente". Por tanto, los programas iterativos que usan pilas aparecen más naturales en la forma postfija. (Desde luego es posible escribir programas recursivos o no recursivos para ambas formas. Estamos hablando de la motivación exclusivamente, o sea, de lo que al inicio se antoja más natural.)

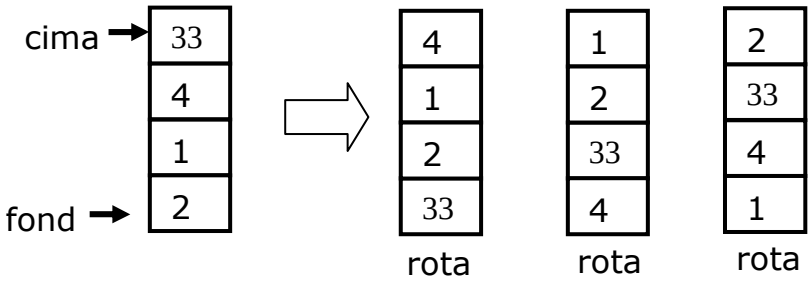
Si queremos evaluar una expresión en la forma postfija, precisa recordar los operandos hasta que su operador se encuentre a la postre un poco después. La manera natural de recordarlos consiste en ponerlos en una pila. Y luego, cuando se encuentra el primer operador que debe efectuarse, hallara sus operandos en la cima de la pila. Si vuelve a colocar en ella su resultado, este estará en el sitio correcto para ser el operando de un operador posterior. Una vez finalizada la evaluación, el resultado final será el único valor sobre la pila. De ese modo, obtenemos un procedimiento para evaluar una expresión postfija.

En este momento debemos notar una diferencia significativa entre las expresiones postfija y prefija. En el procedimiento prefijo no se necesita verificar explícitamente que se haya llegado al final de una expresión, pues la expresión entera constituirá automáticamente el operando (u operandos) del primer operador. Sin embargo, leyendo una expresión postfija de izquierda a derecha, podemos hallar subexpresiones que son, por si mismas, expresiones postfijas legitimas.

Robert L. Kruse (1988) *Estructura de Datos y Diseño de Programas.* México D.F. Prentice-Hall Hispanoamericana S.A. p. 320.

Autoevaluación

1. En una pila de tamaño 4 se quiere rotar los elementos de tal manera como se observa en el ejemplo donde se han hecho tres rotaciones:



Basándonos en el programa para una pila implementado por listas en la sección 1.5 del capítulo Listas y Colas se ha creado el método rotar:

```
void rotarPila(PILA &p){
      PILA aux1(4),aux2(4);
      while(!p.pilaVacia())
         [                          ]
      if(!aux1.pilaVacia())
         [                          ]
      while(!aux1.pilaVacia())
         p.meterPila(aux1.sacarPila());
      if(!aux2.pilaVacia())
         [                           ]
   }
```

Diga usted que alternativa es la que tiene las instrucciones correctas y que reemplazando en el espacio entre corchetes produce la rotacion:

a) aux1.meterPila(p.sacarPila());
   aux2.meterPila(aux1.sacarPila());
   p.meterPila(aux2.sacarPila());

b) p.meterPila(aux2.sacarPila());
   aux2.meterPila(aux1.sacarPila());
   aux1.meterPila(p.sacarPila());

c) aux2.meterPila(p.sacarPila());
   aux1.meterPila(aux1.sacarPila());
   p.meterPila(aux2.sacarPila());

d) ninguna

2. En el programa para una pila implementada por listas se ha cambiado el código del método meterPila por el siguiente:

```
void meterPila(int dato){
        PILA *q,*ptr=new  PILA;
        ptr->numero = dato;
        if(raiz==NULL)raiz=ptr;
        else { q=raiz;
              while(q->sig!=NULL)q=q->sig;
```

```
                q->sig=ptr;
            }
            ptr->sig=NULL;
            i++;
    }
```

¿Cuál dato es el que sale al hacer una operación sacarPila, una vez que se ha ingresado la siguiente secuencia: 2, 3, 4, y 5.

a) 2 b) 5 c) 3 d) ninguna

3. Para Con las instrucciones de abajo, ordenándolas adecuadamente, se pueda crear una cola con los datos que se sacan cada vez de una pila

```
A:  col.meterCola(dato);
B:  dato=pil.sacarPila();
C:  }else cout<<"pila vacia ";
D:  if (!pil.pilaVacia()){
E:   cout<<"\n sale= "<<dato;
```

a) ADECB b) BADCE c) DBEAC d) ninguna

4. Se ha creado el método revisarSecuencia que verifica para una cantidad de números de un mismo valor, existan la misma cantidad de números de igual valor. Por ejemplo secuencias validas son:

```
1 1 1 2 2 2
1 1 2 2
```

Secuencia no valida

```
1 2 2
2 2 1
```

Abajo en la función coloque en los corchetes en blanco las instrucciones de la alternativa que resolverá el problema

```
void revisarSecuencia(PILA &p){
      int x,y;
      PILA aux(10);
      while(!p.pilaVacia()){
          x=p.sacarPila();
         if(!p.pilaVacia()){
            [                          ]
            if(x==y){
               p.meterPila(y);
               [                        ]
            }
            else {
               while(!aux.pilaVacia())
               [                            ]
            }
         }
         else cout<<"\n error en la secuencia";
      }
   }
```

a) p.meterPila(aux.sacarPila());
y=p.sacarPila();
aux.meterPila(x);

b) y=p.sacarPila();
aux.meterPila(x);
p.meterPila(aux.sacarPila());

c) aux.meterPila(x);
y=p.sacarPila();
p.meterPila(aux.sacarPila());

d) ninguna

Claves: 1:d; 2:b; 3:c; 4:d;

Enlaces

http://www.davidparedes.es/2009/01/02/tipos-abstractos-de-datos-tad/
http://www.calcifer.org/documentos/librognome/glib-lists-queues.html
www.lcc.uma.es/~pscp/doc/colas.doc
http://pdf-search-engine.com/pilas-colas-listas-enlazadas-pdf.html
http://fcc98.tripod.com/tutores/ed1/ed1.html#CONTE

Bibliografía

Tenenbaum A. M., Langsam Y., Augenstein, M.A., (1993) *Estructura de Datos en C*, 2ª. Ed., México D.F., Prentice Hall Hispanoamericana S.A. Cap. 4: Colas y Listas pp. 174-248.

Cairó O., Guardati M.C. S., (2002) *Estructura de Datos*, 2ª. Ed. México D.F., Mc Graw-Hill / Interamericana Editores S.A. Cap. 3: Pilas y Colas pp. 126-229.

Kruse, Robert L. (1988) *Estructura de Datos y Diseño de Programas.* México D.F. Prentice-Hall Hispanoamericana S.A. Cap. 4: Pilas pp. 130-158.

Allen Weiss, M., (1995) *Estructura de Datos y Algoritmos*, México D.F., Addison-Wesley Iberoamericana. Cap. 3: Pilas y colas pp.124-155.