Estructuras de datos Colas y Pilas

COLAS:

Las colas son estructuras de datos en las que el primer elemento de información en ingresar es el primero en salir, por lo que se las conoce como listas FIFO (por las siglas en inglés First In – First Out).

Son muy utilizadas en informática, para almacenar datos que necesiten ser procesados según el orden de llegada. Por ejemplo, en un sistema operativo los trabajos de impresión se ordenan en una Cola, y son procesados de acuerdo a su ubicación en ella.

Existen diversas implementaciones de las colas, que responden a necesidades específicas de procesamiento de información. Puede haber colas circulares, dobles, con prioridad, etc.

Operaciones básicas sobre colas

Las operaciones básicas en una cola son las que permiten agregar y sacar elementos de información.

Agregar: se agrega un elemento de información al final de la cola.

Sacar: se saca de la cola, para su procesamiento, el elemento de información más "viejo", es decir, el primero en ser ingresado. Independientemente de la implementación que se haga mediante programación de la Cola, la operación Sacar es destructiva: el elemento de información ya no será accesible. Para quien la utilice debe verse como que no "existe", que el elemento de información ya no forma parte de la estructura que lo contenía.

Adicionalmente, puede necesitarse una operación que permita saber si la cola tiene espacio para almacenar un nuevo elemento (en caso que tenga una longitud fija), o si está vacía, esto es, no tiene ningún elemento de información almacenado.

Implementación de las colas

Una cola puede ser estática o dinámica, de acuerdo a si tiene una longitud fija (sólo admite *n* elementos de información) o si permite un crecimiento indeterminado de la cantidad de componentes. Una cola estática puede construirse a partir de un vector; una dinámica en cambio suele implementarse mediante una lista enlazada.

En el siguiente ejemplo se implementa mediante una clase una cola sencilla de longitud fija que sólo admite que se ingresen y se saquen una cantidad predefinida de números enteros positivos. Con fines didácticos se agrega un método que muestra los componentes de la cola. Se supone que los valores de ingreso aceptados son enteros positivos.

```
#include<iostream>

class Cola
        {
```

```
        private:
                int *p;
                int pcio, fin, tam;
        public:
                Cola (int);
                void Agregar(int);
                int Sacar( );
                void Mostrar( );
                ~Cola( );
        };

Cola::Cola (int cant=1)
        {
        tam=cant;
        pcio=fin=0;
        p=new int [tam];
        if (p= =NULL) exit(1);
        }

void Cola::Agregar(int info)
        {
        if(fin= =tam)
                cout<<"Cola llena";
        else
                {
                p[fin]=info;
                fin++;
                }
        }

int Cola::Sacar( )
        {
        if(pcio= = fin)
                {
                cout<<"Cola vacía";
                return 0;
                }
        else
                return p[pcio++];
        }

void Cola::Mostrar( )
        {
        for(int i=pcio;i<fin;i++)
                cout<<p[i];
        }

Cola::~Cola( ) {delete p;};

int main( )
```

```
{
Cola a(5);
int i, info;
for(i=0;i<5;i++)
        {
        cin>>info;
        a.Agregar(info);
        }
for(i=0;i<5;i++)
        {
        cout<<a.Sacar( ) ;
        a.Mostrar( ) ;
        system(« pause » );
        }
return 0 ;
}
```

En el ejemplo el tamaño de la cola se determina en el momento de declarar el objeto a. Mediante un for se la carga, y luego se la descarga y se muestra los elementos que quedaron en la cola.

En la clase Cola se definen como propiedades el puntero sobre el cual se pide luego memoria para construir el vector (ver el constructor de la clase), y 3 variables enteras que se utilizan de la siguiente manera:
tam: almacena el valor correspondiente a la cantidad de elementos de información. Permanece fijo.
pcio: indica la posición del primer elemento ingresado. A medida que se van sacando elementos de la cola se lo incrementa (ver método Sacar( )), de manera que siempre indica el elemento más “viejo”.
fin: indica la posición del próximo elemento a cargar. Cuando se agrega un elemento, se lo incrementa (ver método Agregar()).

En el método Agregar( ), antes de permitir la carga se analiza si la cola tiene espacio libre. Para ello se compara el valor de fin y el de tam. Si ambos son iguales, significa que la cola está llena.

En el método Sacar( ), antes de devolver el elemento de información se analiza si la cola está vacía. Para eso se compara el valor de pcio y el de fin. Si ambos son iguales, significa que no hay elementos para sacar.

Mediante el destructor se libera la memoria solicitada sobre p.

Resulta fácil comprender que, si bien en el ejemplo el elemento de información es sólo un número, con unas pocas modificaciones se podría hacer que contenga cualquier conjunto de campos.

Ejercicios:

1) La clase del ejemplo sólo admite una cantidad *n* predefinida de componentes; una vez ingresados esos *n* elementos no podrán ser ingresados nuevos, a pesar de que se saque de ella uno o más componentes. Modificar la clase Cola del

ejemplo, para que almacene una cantidad fija de números enteros, pero que admita nuevos ingresos si en ella hay espacio (una vez llena se sacaron uno o más elementos).

2) Hacer una clase Cola que tenga como elemento de información objetos de la clase Articulo, usado durante el año como ejemplo.

3) Hacer una clase Cola que se defina como de tamaño fijo (mínimo), pero que crezca en caso que se quieran agregar más elementos. Tener en cuenta que si bien no es posible agregar componentes a un vector, si éste ha sido creado dinámicamente, se puede devolver la memoria solicitada, y luego pedir otra vez memoria sobre el mismo puntero, pero en este caso sumando una componente a las que tenía anteriormente (ver como hacer para no perder los datos que contenía la cola).

PILAS:

La pila es una estructura con numerosas analogías en la vida real, una pila de platos, una pila de documentos, una pila de monedas. A diferencia de la cola, su comportamiento es tal que el primero que ingresa es el último que sale, o sea LIFO (Last In – First Out). Tanto la operación de inserción (agregar) o eliminar (sacar) se hacen sobre un mismo extremo, generalmente llamado tope.

Las consideraciones sobre las operaciones básicas sobre pilas, así como su implementación son análogas a las hechas en relación a las colas. La diferencia con la cola es que la operación sacar se hace sobre el elemento de información mas “nuevo”, es decir, el último ingresado.

A continuación se presenta una implementación de una pila sencilla en una clase. Se supone que los valores aceptados de información son enteros positivos.

```
#include<iostream>

class Pila
        {
        private:
                int *p;
                int tope, tam;
        public:
                Pila (int);
                void Agregar(int);
                int Sacar( );
                void Mostrar( );
                ~Pila( );
        };

Pila::Pila (int cant=1)
```

```
        {
        tam=cant;
        tope=0;
        p=new int [tam];
        if (p= =NULL) exit(1);
        }

void Pila::Agregar(int aux)
        {
        if(tope= =tam)
                cout<<"Pila llena";
        else
                {
                p[tope]=aux;
                tope++;
                }
        }

int Pila::Sacar( )
        {
        if(tope= = 0)
                {
                cout<<"Pila vacía";
                return 0;
                }
        else
                {
                tope--;
                return p[tope];
                }
        }

void Pila::Mostrar( )
        {
        for(int i=tope-1;i>=0;i--)
                cout<<p[i];
        }

Pila::~Pila( ) {delete p;};

int main( )
        {
        Pila a(5);
        int i, info;
        for(i=0;i<5;i++)
                {
                cin>>info;
                a.Agregar(info);
                }
        for(i=0;i<5;i++)
```

```
        {
        cout<<a.Sacar( ) ;
        a.Mostrar( ) ;
        system(« pause » );
        }
return 0 ;
}
```

En el ejemplo el tamaño de la pila se determina en el momento de declarar el objeto a. Mediante un for se la carga, y luego se la descarga y se muestra los elementos que quedaron en la pila.

En la clase Pila se definen como propiedades el puntero sobre el cual se pide luego memoria para construir el vector (ver el contructor de la clase), y 2 variables enteras que se utilizan de la siguiente manera:
tam: almacena el valor correspondiente a la cantidad de elementos de información. Permanece fijo.
tope: indica la posición del próximo elemento a cargar. Cuando se agrega un elemento, se lo incrementa (ver método Agregar()).
No es necesario indicar, como en la clase Cola, el principio, ya que se sabe que es 0.

En el método Agregar( ), antes de permitir la carga se analiza si la pila tiene espacio libre. Para ello se compara el valor de tope y el de tam. Si ambos son iguales, significa que la pila está llena.

En el método Sacar( ), antes de devolver el elemento de información se analiza si la pila está vacía. Para eso se compara el valor de tope. Si es 0, significa que no hay elementos para sacar.

Mediante el destructor se libera la memoria solicitada sobre p.

Resulta fácil comprender que, si bien en el ejemplo el elemento de información es sólo un número, con unas pocas modificaciones se podría hacer que contenga cualquier conjunto de campos.

Ejercicios:

1) La clase del ejemplo admite una cantidad *n* predefinida de componentes; una vez ingresados esos *n* elementos, podrán ser ingresados nuevos si se saca de ella uno o más componentes. Modificar la clase Pila del ejemplo, para que almacene una cantidad fija de números enteros, pero que no admita nuevos ingresos si ya se ha ingresado la cantidad predefinidad de elementos de información, a pesar de que exista espacio (una vez llena se sacaron uno o más elementos).

2) Hacer una clase Pila que tenga como elemento de información objetos de la clase Artículo, usado durante el año como ejemplo.

3) Hacer una clase Pila que se defina como de tamaño fijo (mínimo), pero que crezca en caso que se quieran agregar más elementos. Tener en cuenta que si

bien no es posible agregar componentes a un vector, si éste ha sido creado dinámicamente, se puede devolver la memoria solicitada, y luego pedir otra vez memoria sobre el mismo puntero, pero en este caso sumando una componente a las que tenía anteriormente (ver como hacer para no perder los datos que contenía la pila).

4) Hacer un programa en el que se cargue una pila, y luego se descargue sobre una cola de igual longitud.