

Universidad de Guadalajara

Centro Universitario de Ciencias Exactas e Ingenierías



IL365 - Estructura de Datos - D01

Actividad de Aprendizaje #11

La Pila y la Cola, Implementación Dinámica

Alumna: Cervantes Araujo Maria Dolores

Código: 217782452

Fecha de Elaboración: 17 abril de 2023



Autoevaluación			
Concepto	Si	No	Acumulación
Bajé el trabajo de internet o alguien me lo pasó (aunque sea de forma parcial)	-100 pts	0 pts	0
Incluí el código fuente en formato de texto (sólo si funciona cumpliendo todos los requerimientos)	+25pts	0 pts	25
Incluí las impresiones de pantalla (sólo si funciona cumpliendo todos los requerimientos)	+25pts	0 pts	25
Incluí una portada que identifica mi trabajo (nombre, código, materia, fecha, título)	+25 pts	0 pts	25
Incluí una descripción y conclusiones de mi trabajo	+25 pts	0 pts	25
Suma:			100

Introducción:

En esta semana, retomamos el tema de la pila y cola dinámica previamente explicado en clase para ahora pasar a la codificación. Una vez que ya trabajamos con listas dinámicas, ahora llegó el turno de la implementación para la pila y la cola, usando un método de simplemente ligado para la pila y uno doblemente ligado circular con encabezado Dummy para el caso de la cola. Se retomó la práctica 4 de conversión de operaciones infijas a operaciones posfijas.

El uso de ambos métodos dinámicos resultó sencillo una vez que lo trabajamos con las listas, sin embargo, se tuvieron que hacer adaptaciones en la pila y la cola para seguir detectando los elementos para el push, pop, queue y dequeue, sin realizar ningún otro cambio en las demás clases del algoritmo.



Código Fuente:

Stack.hpp

```
#ifndef STACK_HPP_INCLUDED
#define STACK_HPP_INCLUDED
#include <iostream>
#include "LolException.hpp"

/// DEFINICION
template<class T>
class Stack {
    private:
        class Node {
            private:
                T data;
                Node* next;
            public:
                Node();
                Node(const T&);

                T getData() const;
                Node* getNext() const;

                void setData(const T&);
                void setNext(Node*);
        };

        Node* anchor;
        void copyAll(const Stack&);
        void deleteAll();

    public:
        Stack();
        Stack(const Stack&);
        ~Stack();

        bool isEmpty() const;

        void push(const T&);
        T pop();
        T getTop();

        Stack<T>& operator = (const Stack&);
};

/// IMPLEMENTACION NODO
template <class T>
Stack<T>::Node::Node() : next(nullptr) { }

template <class T>
Stack<T>::Node::Node(const T& m) : data(m), next(nullptr) { }
```



```
template <class T>
T Stack<T>::Node::getData() const {
    return data;
}

template <class T>
typename Stack<T>::Node* Stack<T>::Node::getNext() const {
    return next;
}

template <class T>
void Stack<T>::Node::setData(const T& m) {
    data = m;
}

template <class T>
void Stack<T>::Node::setNext(Node* pos) {
    next = pos;
}

///IMPLEMENTACION STACK
using namespace std;
template<class T>
Stack<T>::Stack() : anchor(nullptr) { }

template<class T>
Stack<T>::Stack(const Stack& obj) : anchor(nullptr) {
    copyAll(obj);
}

template<class T>
Stack<T>::~~Stack() {
    deleteAll();
}

template<class T>
bool Stack<T>::isEmpty() const {
    return anchor == nullptr;
}

template<class T>
void Stack<T>::push(const T& obj) {
    Node* aux(new Node(obj));

    if(aux==nullptr) {
        throw Exception("Desbordamiento de datos.");
    }
    aux->setNext(anchor);
    anchor = aux;
    //pila[++top] = obj;
}
```



```
template<class T>
T Stack<T>::pop() {
    if(anchor == nullptr) {
        throw Exception("Insuficiencia de datos.");
    }

    T result(anchor->getData());
    Node* aux(anchor);
    anchor = anchor->getNext();

    delete aux;
    return result;
    //return pila[top--];
}
```

```
template<class T>
T Stack<T>::getTop() {
    if(isEmpty()) {
        throw Exception("Insuficiencia de datos.");
    }
    return anchor->getData();
}
```

```
template<class T>
Stack<T>& Stack<T>::operator = (const Stack& obj) {
    deleteAll();
    copyAll(obj);
    return *this;
}
```

```
template<class T>
void Stack<T>::copyAll(const Stack<T>& obj) {
    Node* aux(obj.anchor);
    Node* last(nullptr);
    Node* newNode;

    while(aux != nullptr) {
        newNode = new Node(aux->getData());

        if(last == nullptr) {
            anchor = newNode;
        }
        else {
            last->setNext(newNode);
        }
        last = newNode;
        aux = aux->getNext();
    }
}
```

```
template<class T>
void Stack<T>::deleteAll() {
    Node* aux;
```



```
while(anchor != nullptr) {  
    aux = anchor;  
    anchor = anchor->getNext();  
    delete aux;  
}
```

```
#endif // STACK_HPP_INCLUDED
```

Queue.hpp

```
#ifndef QUEUE_HPP_INCLUDED  
#define QUEUE_HPP_INCLUDED  
#include <iostream>  
#include "LolException.hpp"
```

```
/// DEFINICION
```

```
template<class A>
```

```
class Queue {
```

```
private:
```

```
    class Node {
```

```
        private:
```

```
            A* dataPtr;
```

```
            Node* prev;
```

```
            Node* next;
```

```
        public:
```

```
            class Exception: public std::exception {
```

```
                private:
```

```
                    std::string msg;
```

```
                public:
```

```
                    explicit Exception(const char* message): msg(message) {}
```

```
                    explicit Exception(const std::string& message):
```

```
msg(message) {}
```

```
                    virtual ~Exception() throw() {}
```

```
                    virtual const char* what() const throw() {
```

```
                        return msg.c_str();
```

```
                    }
```

```
            };
```

```
        Node();
```

```
        Node(const A&);
```

```
        ~Node();
```

```
        A* getDataPtr() const;
```

```
        A getData() const;
```

```
        Node* getPrev() const;
```

```
        Node* getNext() const;
```

```
        void setDataPtr(const A*);
```

```
        void setData(const A&);
```

```
        void setPrev(Node*);
```



```
        void setNext (Node*);  
    };  
  
    Node* header;  
    void copyAll (const Queue<A>&);  
    void deleteAll ();  
  
public:  
    Queue ();  
    Queue (const Queue& );  
    ~Queue ();  
  
    bool isEmpty ();  
    void enqueue (const A& );  
    A dequeue ();  
    A getFront ();  
  
    Queue<A>& operator = (const Queue&);  
};  
  
/// IMPLEMENTACION NODO  
using namespace std;  
template <class A>  
Queue<A>::Node::Node () : dataPtr (nullptr), prev (nullptr), next (nullptr) { }  
  
template <class A>  
Queue<A>::Node::Node (const A& m) : dataPtr (new A(m)), prev (nullptr), next (nullptr)  
{  
    if (dataPtr == nullptr) {  
        throw Exception ("Memoria insuficiente, se creara un nodo");  
    }  
}  
  
template <class A>  
Queue<A>::Node::~~Node () {  
    delete dataPtr;  
}  
  
template <class A>  
A* Queue<A>::Node::getDataPtr () const {  
    return dataPtr;  
}  
  
template <class A>  
A Queue<A>::Node::getData () const {  
    if (dataPtr == nullptr) {  
        throw Exception ("Dato inexistente, getData");  
    }  
    return *dataPtr;  
}  
  
template <class A>  
typename Queue<A>::Node* Queue<A>::Node::getPrev () const {
```



```
    return prev;
}

template <class A>
typename Queue<A>::Node* Queue<A>::Node::getNext() const {
    return next;
}

template <class A>
void Queue<A>::Node::setDataPtr(const A* pos) {
    dataPtr = pos;
}

template <class A>
void Queue<A>::Node::setData(const A& e) {
    if(dataPtr == nullptr) {
        if((dataPtr = new A(e)) == nullptr) {
            throw Exception("Memoria no disponible, setData");
        }
    }
    else {
        *dataPtr = e;
    }
}

template <class A>
void Queue<A>::Node::setPrev(Node* pos) {
    prev = pos;
}

template <class A>
void Queue<A>::Node::setNext(Node* pos) {
    next = pos;
}

///IMPLEMENTACION QUEUE
using namespace std;
template<class A>
void Queue<A>::copyAll(const Queue<A>& obj) {
    Node* aux(obj.header->getNext());
    Node* newNode;

    while(aux != obj.header) {
        try {
            if((newNode = new Node(aux->getData())) == nullptr) {
                throw Exception("Memoria no disponible, copyAll");
            }
        }
        catch(typename Node::Exception ex) {
            throw Exception(ex.what());
        }

        newNode->setPrev(header->getPrev());
    }
}
```




```
newNode->setNext(header);

header->getPrev()->setNext(newNode);
header->setPrev(newNode);

aux = aux->getNext();
}
}

template<class A>
Queue<A>::Queue() : header(new Node) {
    if(header == nullptr) {
        throw Exception("Memoria no disponible");
    }

    header->setPrev(header);
    header->setNext(header);
}

template<class A>
Queue<A>::Queue(const Queue& obj) : Queue() {
    copyAll(obj);
}

template<class A>
Queue<A>::~~Queue() {
    deleteAll();
    delete header;
}

template<class A>
bool Queue<A>::isEmpty() {
    return (header->getNext() == header);
}

template<class A>
void Queue<A>::enqueue(const A& obj) {
    Node* aux;
    try {
        if((aux = new Node(obj)) == nullptr) {
            throw Exception("Memoria no disponible, EnqueuePostFija");
        }
    }
    catch(typename Node::Exception ex) {
        throw Exception(ex.what());
    }
    aux->setPrev(header->getPrev());
    aux->setNext(header);

    header->getPrev()->setNext(aux);
    header->setPrev(aux);
}
```



```
template<class A>
A Queue<A>::dequeue() {
    if(isEmpty()) {
        throw Exception("Insuficiencia de datos.");
    }

    A result(header->getNext()->getData());
    Node* aux(header->getNext());

    aux->getPrev()->setNext(aux->getNext());
    aux->getNext()->setPrev(aux->getPrev());

    delete aux;
    return result;
}

template<class A>
A Queue<A>::getFront() {
    if(isEmpty()) {
        throw Exception("Insuficiencia de datos.");
    }
    return header->getNext()->getData();
}

template<class A>
Queue<A>& Queue<A>::operator = (const Queue& obj) {
    copyAll(obj);
    return *this;
}

template <class A>
void Queue<A>::deleteAll() {
    Node* aux;

    while(header->getNext() != header) {
        aux = header->getNext();
        header->setNext(aux->getNext());
        delete aux;
    }
    header->setPrev(header);
}

#endif // QUEUE_HPP_INCLUDED
```



QueuePostFija.hpp

```
#ifndef QUEUEPOSFIJO_HPP_INCLUDED
#define QUEUEPOSFIJO_HPP_INCLUDED
#include <iostream>
#include "LolException.hpp"

/// DEFINICION
template<class A>
class QueuePosfijo {
private:
    class Node {
private:
        A* dataPtr;
        Node* prev;
        Node* next;

public:
        class Exception: public std::exception {
private:
            std::string msg;
public:
            explicit Exception(const char* message): msg(message) {}
            explicit Exception(const std::string& message):
                msg(message) {}

            virtual ~Exception() throw() {}
            virtual const char* what() const throw() {
                return msg.c_str();
            }
        };

        Node();
        Node(const A&);
        ~Node();

        A* getDataPtr() const;
        A getData() const;
        Node* getPrev() const;
        Node* getNext() const;

        void setDataPtr(const A*);
        void setData(const A&);
        void setPrev(Node*);
        void setNext(Node*);
    };

    Node* header;
    void copyAll(const QueuePosfijo<A>&);
    void deleteAll();

public:
    QueuePosfijo();
    QueuePosfijo(const QueuePosfijo&);
};
```



```
~QueuePosfijo();

bool isEmpty();
void enqueue(const A& );
A dequeue();
A getFront();

QueuePosfijo<A>& operator = (const QueuePosfijo&);
};

///IMPLEMENTACION NODO
using namespace std;
template <class A>
QueuePosfijo<A>::Node::Node() : dataPtr(nullptr), prev(nullptr), next(nullptr) { }

template <class A>
QueuePosfijo<A>::Node::Node(const A& m) : dataPtr(new A(m)), prev(nullptr),
next(nullptr) {
    if(dataPtr == nullptr) {
        throw Exception("Memoria insuficiente, se creara un nodo");
    }
}

template <class A>
QueuePosfijo<A>::Node::~~Node() {
    delete dataPtr;
}

template <class A>
A* QueuePosfijo<A>::Node::getDataPtr() const {
    return dataPtr;
}

template <class A>
A QueuePosfijo<A>::Node::getData() const {
    if(dataPtr == nullptr) {
        throw Exception("Dato inexistente, getData");
    }
    return *dataPtr;
}

template <class A>
typename QueuePosfijo<A>::Node* QueuePosfijo<A>::Node::getPrev() const {
    return prev;
}

template <class A>
typename QueuePosfijo<A>::Node* QueuePosfijo<A>::Node::getNext() const {
    return next;
}

template <class A>
void QueuePosfijo<A>::Node::setDataPtr(const A* pos) {
```



```
dataPtr = pos;
}

template <class A>
void QueuePosfijo<A>::Node::setData(const A& e) {
    if(dataPtr == nullptr) {
        if((dataPtr = new A(e)) == nullptr) {
            throw Exception("Memoria no disponible, setData");
        }
    }
    else {
        *dataPtr = e;
    }
}

template <class A>
void QueuePosfijo<A>::Node::setPrev(Node* pos) {
    prev = pos;
}

template <class A>
void QueuePosfijo<A>::Node::setNext(Node* pos) {
    next = pos;
}

/// IMPLEMENTACION QUEUEPOSTFIJO
using namespace std;
template<class A>
void QueuePosfijo<A>::copyAll(const QueuePosfijo<A>& obj) {
    Node* aux(obj.header->getNext());
    Node* newNode;

    while(aux != obj.header) {
        try {
            if((newNode = new Node(aux->getData())) == nullptr) {
                throw Exception("Memoria no disponible, copyAll");
            }
        }
        catch(typename Node::Exception ex) {
            throw Exception(ex.what());
        }

        newNode->setPrev(header->getPrev());
        newNode->setNext(header);

        header->getPrev()->setNext(newNode);
        header->setPrev(newNode);

        aux = aux->getNext();
    }
}
```



```
template<class A>
QueuePosfijo<A>::QueuePosfijo() : header(new Node) {
    if(header == nullptr) {
        throw Exception("Memoria no disponible");
    }

    header->setPrev(header);
    header->setNext(header);
}

template<class A>
QueuePosfijo<A>::QueuePosfijo(const QueuePosfijo& obj) : QueuePosfijo() {
    copyAll(obj);
}

template<class A>
QueuePosfijo<A>::~~QueuePosfijo() {
    deleteAll();
    delete header;
}

template<class A>
bool QueuePosfijo<A>::isEmpty() {
    return (header->getNext() == header);
}

template<class A>
void QueuePosfijo<A>::enqueue(const A& obj) {
    Node* aux;
    try {
        if((aux = new Node(obj)) == nullptr) {
            throw Exception("Memoria no disponible, EnqueuePostFija");
        }
    }
    catch(typename Node::Exception ex) {
        throw Exception(ex.what());
    }
    aux->setPrev(header->getPrev());
    aux->setNext(header);

    header->getPrev()->setNext(aux);
    header->setPrev(aux);
}

template<class A>
A QueuePosfijo<A>::dequeue() {
    if(isEmpty()) {
        throw Exception("Insuficiencia de datos.");
    }

    A result(header->getNext()->getData());
    Node* aux(header->getNext());
```



```
aux->getPrev()->setNext(aux->getNext());
aux->getNext()->setPrev(aux->getPrev());

delete aux;
return result;
}

template<class A>
A QueuePosfijo<A>::getFront() {
    if(isEmpty()) {
        throw Exception("Insuficiencia de datos.");
    }
    return header->getNext()->getData();
}

template<class A>
QueuePosfijo<A>& QueuePosfijo<A>::operator = (const QueuePosfijo& obj) {
    copyAll(obj);
    return *this;
}

template <class A>
void QueuePosfijo<A>::deleteAll() {
    Node* aux;

    while(header->getNext() != header) {
        aux = header->getNext();
        header->setNext(aux->getNext());
        delete aux;
    }
    header->setPrev(header);
}

#endif // QUEUEPOSFIJO_HPP_INCLUDED
```



Menu.hpp

```
#ifndef MENU_HPP_INCLUDED
#define MENU_HPP_INCLUDED
#include "QueuePosfijo.hpp"
#include "Queue.hpp"
#include "Stack.hpp"
#include <iostream>
#include <string.h>

class Menu {
private:
    int op;
    Stack<char> Spila;
    Queue<char> colaInserc;
    QueuePosfijo<char> colaOut;
    char aux;

public:
    Menu();
    Menu(const Menu& );

    void convertinfijo_Postfijo();
    bool isOperador(const char&);
    int precedence(const char&);
    Menu& operator = (const Menu& );
};

#endif // MENU_HPP_INCLUDED
```

Menu.cpp

```
#include "Menu.hpp"

using namespace std;
Menu::Menu() {
}

Menu::Menu(const Menu& m) {
    colaInserc = m.colaInserc;
    colaOut = m.colaOut;
    Spila = m.Spila;
}

void Menu::convertinfijo_Postfijo() {
    char c[150];
    char aux;
    int i=0, tam, oper;

    cout<<"\n--> Digita una operacion"<<endl;
    cin.getline(c, 150, '\n');
```




```
tam=strlen(c);
while(i<tam) {
    colaInserc.enqueue(c[i]); ///Inserta la operacion inija
    i++;
}
while(!colaInserc.isEmpty()) {

    if(isOperador(colaInserc.getFront())) {
        oper=precedence(colaInserc.getFront());
        ///Si no es parentesis de apertura && parentesis menor a precedente
        apila el operador
        while(!Spila.isEmpty() && (aux=Spila.getTop()!='(') && oper<=
precedence(Spila.getTop())) {
            colaOut.enqueue(Spila.pop()); ///Se apila el digito en la cola del
            resultado y se eliminan los operadores
        }
        Spila.push(colaInserc.dequeue()); ///Se recorren las posiciones de la
        cola
    }
    else if(colaInserc.getFront()=='(') {
        Spila.push(colaInserc.dequeue());
    }

    ///Si parentesis de cierre:
    /// o que la pila este vacia desapilar operadores y pasarlos al resultado
    hasta enocntrar un parentesis de apertura
    else if(colaInserc.getFront()==')') {
        while((aux=Spila.getTop()!='(') {
            colaOut.enqueue(aux);
            Spila.pop();          ///Y desapilar el parentesis de apertura
        }
        colaInserc.dequeue();
        Spila.pop();
    }
    else {
        colaOut.enqueue(colaInserc.dequeue());
    }
}
while(!Spila.isEmpty()) {
    colaOut.enqueue(Spila.pop());
}
while(!colaOut.isEmpty()) {
    cout<<colaOut.dequeue();
}
}

///Si se encuentran operadores.
///Desapilar operadores y pasarlos al resultado mientras estos sean de mayor o
igual precedencia que el que se esta leyendo
///Y no se encuentre con un parentesis de apertura o la pila este vacia
///apilar el operador leído Y Si es un operando/numero: pasarlo a la cola del
```



resultado

```
bool Menu::isOperador(const char&c) {  
    char operadores[5]= {'+', '-', '*', '/', '^'};  
    bool val(false);  
    for(int i=0; i<5; i++) {  
        if(c==operadores[i]) {  
            val=true;  
        }  
    }  
    return val;  
}
```

///PRIORIDAD DE OPERADORES

```
int Menu::precedence(const char&c) {  
    return (c=='+' || c=='-')? 1:  
           (c=='*' || c=='/')? 2:  
           (c=='^')? 3:0;  
}
```

```
Menu& Menu::operator = (const Menu& m) {  
    Spila = m.Spila;  
    colaInserc = m.colaInserc;  
    colaOut = m.colaOut;  
  
    return *this;  
}
```

LolException.hpp

```
#ifndef LOLEXCEPTION_HPP_INCLUDED  
#define LOLEXCEPTION_HPP_INCLUDED  
#include <exception>  
#include <iostream>  
class Exception: public std::exception {  
  
    private:  
        std::string msg;  
    public:  
        explicit Exception(const char* message): msg(message) {}  
        explicit Exception(const std::string& message): msg(message) {}  
        virtual ~Exception() throw() {}  
        virtual const char* what() const throw() {  
            return msg.c_str();  
        }  
};  
  
#endif // LOLEXCEPTION_HPP_INCLUDED
```



Main.cpp

```
#include <iostream>
#include "Menu.hpp"
int main() {
    Menu start;
    start.convertinfijo_Postfijo();
    return 0;
}
```

Impresiones de Pantalla:

```
"C:\Users\cerva\Escritorio\F.Prog\Estructura de datos\Actividad11\bin\Debug\Actividad11.exe"

--> Digita una operacion
((((((A+B)*C)-D)^E)/F)+G)*H
AB+C*D-E^F/G+H*
Process returned 0 (0x0)   execution time : 143.140 s
Press any key to continue.
```

```
"C:\Users\cerva\Escritorio\F.Prog\Estructura de datos\Actividad11\bin\Debug\Actividad11.exe"

--> Digita una operacion
(((6+7-5)^G)-U/R)/4*1
67+5-G^UR/-4/1*
Process returned 0 (0x0)   execution time : 73.202 s
Press any key to continue.
```

```
"C:\Users\cerva\Escritorio\F.Prog\Estructura de datos\Actividad11\bin\Debug\Actividad11.exe"

--> Digita una operacion
((((((AE+45/U)*E)^3*7)-Y8)*(^9)/G+T)-6
AE45U/+E*3^7*Y8-9^*G/T+6-
Process returned 0 (0x0)   execution time : 47.774 s
Press any key to continue.
```

```
"C:\Users\cerva\Escritorio\F.Prog\Estructura de datos\Actividad11\bin\Debug\Actividad11.exe"

--> Digita una operacion
((((87*4)/8+G)^2)/I-6)*R
874*8/G+2^I/6-R*
Process returned 0 (0x0)   execution time : 112.906 s
Press any key to continue.
```



```
"C:\Users\cerva\Escritorio\F.Prog\Estructura de datos\Actividad11\bin\Debug\Actividad11.exe"

--> Digita una operacion
8+9(((E*R/7)-U)^2+5)*T^(((B-Q)/3)*U)+4
89ER*7/U-2^5+TBQ-3/U*^+4+
Process returned 0 (0x0)   execution time : 138.786 s
Press any key to continue.

"C:\Users\cerva\Escritorio\F.Prog\Estructura de datos\Actividad11\bin\Debug\Actividad11.exe"

--> Digita una operacion
7+8(U*R(Y/Q)^3D-MA)+2(/J((-8)^P)*F)
78URYQ/3D^*MA--+2J8-P^/F*+
Process returned 0 (0x0)   execution time : 95.789 s
Press any key to continue.
```

Conclusión

Esta semana se trabajaron pilas y colas dinámicas, para la implementación solo fue necesario realizar unos cambios del trabajo que ya teníamos pasando de estático a dinámico, no me causo mayor complejidad, seguí el algoritmo del pseudo código presentado por el profesor y los errores que llegaron a presentarse fueron por errores de dedo, ya sea que le ponía una letra de más o me faltaba agregar algún carácter; de ahí en más fue una actividad sencilla porque anteriormente había trabajado con lista simplemente ligada y la doblemente ligada con encabezado Dummy por lo que solo traspase la parte del código que era correspondiente al nodo para formar mi base y concluir la actividad.

Puedo concluir, que estas alturas ya es posible identificar los métodos dinámicos y estáticos, así como el modelo que corresponde con las listas simplemente ligadas o doblemente ligadas circular con o sin encabezado, si bien no es un tema complejo, puede llegar a revolver al principio.