ACTIVIDAD 11: **LA PILA Y LA COLA, IMPLEMENTACIÓN DINÁMICA**

ESTUDIANTE: **EFRAIN ROBLES PULIDO**

CODIGO: **221350095**

NOMBRE DE LA MATERIA: **ESTRUCTURAS DE DATOS I**

SECCIÓN: ***D12***

CLAVE: ***I5886***

FECHA: **SABADO 2 DE ABRIL DE 2022**

Tabla de autoevaluación:

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

Introducción:

Después de leer las instrucciones de la plataforma, se reutilizo el programa de la actividad 4, en donde se realizó el convertidor de expresiones de infija a posfija. Pero para esta actividad se modificó las clases Pila y Cola, en donde se utilizan Nodos, y cada uno tiene funcionamientos diferentes para cada clase, como en la pila de forma dinámica se implementó la **LSLLSE,** en donde los nodos están simplemente ligada y es lineal y no tiene un encabezado, siendo el modelo básico para esta clase ya que no es necesario para este tipo de TDA (tipo de dato abstracto), y para la cola de forma dinámica se implementó la **LDLCCED,** en donde los nodos están doblemente ligadas y es circular y aparte tiene el encabezado Dummy (un nodo, pero sin un dato dentro), que está conectado por sus apuntadores siguiente al primer nodo y anterior al último nodo, de la cola, permitiéndonos tener más información de la cola de cuál es su frente y su final, para poder modificarlo apropiadamente, además el atributo a guardar también es un apuntador, para que cada vez que pidamos la información en su método siempre nos de su contenido en esa dirección de memoria.

Una vez estudiado y entendido los temas, después se cambió las clases Pila y Cola de la anterior actividad. Se verifico las instrucciones para esta actividad se obtuvo el mismo resultado de la práctica anterior, pero teniendo un funcionamiento muy diferente, ya que usamos nodos, apuntadores y el almacenamiento de la información se hace de manera dinámica. Finalmente, se le aplico los mismos ejercicios vistos en clase también se le aplico unos diferentes a la actividad anterior.

Código fuente:

```
///Efrain Robles Pulido

#include "Interface.h"
#include <iostream>

int main() {
    Interface myInterface;
    myInterface.mainInterface();
    return 0;
    }

#ifndef INTERFACE_H_INCLUDED
#define INTERFACE_H_INCLUDED

#include "Stack.h"
#include "Queue.h"
#include "Expression.h"
#include <iostream>

class Interface {
    private:
        ///Atributos
        std::string expres;
        Queue<char> firstQueue, secondQueue;
        Stack<char> myStack;

    public:
        ///Constructor
        Interface();
        Interface(Interface&);

        ///Sobrecarga de operador
        Interface& operator = (Interface&);

        ///Metodos
        void convertToPosFix();
        void mainInterface();

    };

#endif // INTERFACE_H_INCLUDED


#include "Interface.h"

using namespace std;

///Contructores
Interface::Interface() {}

Interface::Interface(Interface& i): expres(i.expres),
firstQueue(i.firstQueue), secondQueue(i.secondQueue), myStack(i.myStack)
{}
```

```
///Sobrecarga de operador
Interface& Interface::operator = (Interface& i) {
    expres = i.expres;
    firstQueue = i.firstQueue;
    secondQueue = i.secondQueue;
    myStack = i.myStack;
    }
///Metodos
void Interface::convertToPosFix() {

    cout << "Ingrese expresion con notacion infija: ";
    cin>> expres;

    int i(expres.size());
    while(i >= 0) {
        firstQueue.enqueue(expres[i]);
        i--;
        }

    Expression myExpression(myStack);
    myExpression.getInfix(firstQueue);
    secondQueue = myExpression.convertPosfix();

    cout << "Conversion a notacion posfija: ";
    while(!secondQueue.isEmpty()) {
        cout << secondQueue.dequeue();
        }
    cout << endl << endl;
    }

void Interface::mainInterface() {
    char opc;

    cout<< "\tConvertidor de Notacion de Infija a Posfija"<<endl<<endl;
    do {
        convertToPosFix();

        cout << "Desea convertir otra expresion [S]i/[N]o): ";
        cin >> opc;
        opc = toupper(opc);

        cout << endl;

        }
    while(opc != 'N');
    }

#ifndef EXPRESSION_H_INCLUDED
#define EXPRESSION_H_INCLUDED

#include "Stack.h"
#include "Queue.h"

class Expression {
    private:
        ///Atributo
        Stack<char> *myStack;
```

```
        ///Metodos privados
        int getPrecedence(const char&) const; //Pasamos los operadores
como valores, como la precedencia

        //Determina la precedencia, de la pila a utilizar y la de la cola
infija
        int comparePrecedence(Stack<char>, char)const;

    public:
        ///Contructores
        Expression();
        Expression(Stack<char>&);

        ///Metodos
        void getInfix(Queue<char>&);
        Queue<char> convertPosfix() ;
    };


#endif // EXPRESSION_H_INCLUDED


#include "Expression.h"

using namespace std;

///Metodos privados
int Expression::getPrecedence(const char& operators)const {
    switch (operators) {
        case '+':
        case '-':
            return 1;

        case '/':
        case '*':
            return 2;

        case '^':
            return 3;

        case '(':
            return 4;

        case ')':
            return 5;

        default:
            return 0;
        }
    }

int Expression::comparePrecedence(Stack<char> stackOp, char
operatorOfString)const {
    char stackSymbol;
    int i(0), opPos(0);
```

```
    while(!stackOp.isEmpty()) {
        i++;
        stackSymbol = stackOp.pop();

        if(getPrecedence(stackSymbol) == 4) { //Si el operador de la pila
operador es (
            break;
            }
        if(getPrecedence(operatorOfString) <= getPrecedence(stackSymbol))
{ //Si encuentra de mayor o igual presedencia
            opPos = i;
            }
        }
    if(opPos != 0) {
        return opPos;
        }
    else {
        return -1;
        }
    }

///Constructores
Expression::Expression() {}
Expression::Expression(Stack<char>& s): myStack(&s) {}

///Metodos
void Expression::getInfix(Queue<char>& q) {
    while(!q.isEmpty()) {
        myStack->push(q.dequeue());
        }
    }
Queue<char> Expression::convertPosfix() {
    Queue<char> queueConvert;
    Stack<char> stackOperator;
    int operatorPos(0);

    while(!myStack->isEmpty()) {

        if(getPrecedence(myStack->getTop()) == 0) {//Si un operando
            queueConvert.enqueue(myStack->pop());

            }
        else {//Si NO es un operando

            if(stackOperator.isEmpty()) { //Si la pila del operador esta
vacia
                stackOperator.push(myStack->pop());
                }
            else {
                if(getPrecedence(myStack->getTop()) == 5) { //Si el
operador es un )
                    while(getPrecedence(stackOperator.getTop()) != 4) {
                        queueConvert.enqueue(stackOperator.pop());
                        }

                    stackOperator.pop();//Se elimina el )
                    myStack->pop();
```

```
                    }
                else if(getPrecedence(myStack->getTop()) == 4) {//Si es(,
se apila
                    stackOperator.push(myStack->pop());

                    }
                else if((operatorPos = comparePrecedence(stackOperator,
myStack->getTop())) != -1) { //Si es un operador mayor o igual
precedencia
                    while(operatorPos > 0) {
                        queueConvert.enqueue(stackOperator.pop());
                        operatorPos--;
                        }
                    stackOperator.push(myStack->pop());//Se apila el
operador comparada
                    }
                else {
                    stackOperator.push(myStack->pop());//Se apila el
operador
                    }
                }
            }
        }

    while(!stackOperator.isEmpty()) {//Vacia los operadores de la pila
        queueConvert.enqueue(stackOperator.pop());
        }

    return queueConvert;
    }

#ifndef STACK_H_INCLUDED
#define STACK_H_INCLUDED

#include <string>
#include <exception>

///Definicion de Pila
template <class T>
class Stack {           ///LSLLSE
    private:
        ///Definicion del Nodo
        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*);
            };
```

```
    public:
        typedef Node* Position;

        ///Definicion de Excepcion para la Pila
        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();
                    }
            };

    private:
        ///Atributo
        Position anchor;//Se abre esta area para aprovechar e incializar
lo que tengo anteriormente

        ///Metodos privados
        void copyAll(const Stack&);//Guardara el objeto creado a una
copia
        void deleteAll();//Como ya no es parte del metodo de la pila se
implementara como un metodo privado

    public:
        ///Constructores
        Stack();
        Stack(const Stack&);

        ~Stack(); ///Destructor

        ///Sobrecarga de operador
        Stack& operator = (const Stack&);//No es necesario el <T> porque
para el compilador ya esta todo esta plantillado asi que ya no es
necesario

        ///Metodos para una Pila
        bool isEmpty()const;

        void push(const T&);

        T pop();

        T getTop()const;
    };

///Implementacion
using namespace std;

///del Nodo
template <class T>
Stack<T>::Node::Node():next(nullptr) {}
```

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

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

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

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

template <class T>
void Stack<T>::Node::setNext(Node* p) {
    next = p;
    }
///de la Pila
///Metodos privados
template <class T>
void Stack<T>::copyAll(const Stack<T>& l) {
    Position aux(l.anchor);
    Position last(nullptr);
    Position newNode;

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

        if(newNode == nullptr) {
            throw Exception("Memoria no disponible, copyAll");
            }
        if(last == nullptr) {
            anchor = newNode;
            }
        else {
            last->setNext(newNode);
            }
        last = newNode;
        aux = aux->getNext();
        }
    }
template <class T>
void Stack<T>::deleteAll() {
    Position aux;

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

```cpp
///Constructores
template <class T>
Stack<T>::Stack(): anchor(nullptr) {}

template <class T>
Stack<T>::Stack(const Stack& s): Stack() {
    copyAll(s);
    }

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

///Sobre carga de operador
template <class T>
Stack<T>& Stack<T>::operator =(const Stack<T>&s) {
    deleteAll();
    copyAll(s);
    return *this;
    }

///Metodos publicos de la Pila
template <class T>
bool Stack<T>::isEmpty() const {
    return anchor == nullptr;
    }

template <class T>
void Stack<T>::push(const T& e) {
    Position aux(new Node(e));
    if(aux == nullptr) {
        throw Exception ("Memoria no disponible, push");
        }
    //Religado
    aux->setNext(anchor);
    anchor = aux;
    }

template <class T>
T Stack<T>::pop() {
    if(anchor == nullptr) {
        throw Exception ("Insuficiencia de datos, pop");
        }
      T result (anchor->getData());

    Position aux(anchor);
    anchor = anchor->getNext();

    delete aux;

    return result;
    }
```

```cpp
template <class T>
T Stack<T>::getTop() const {
    if(anchor == nullptr) {
        throw Exception ("Insuficiencia de datos, getTop");
        }
    return anchor->getData();
    }

#endif // STACK_H_INCLUDED

#ifndef QUEUE_H_INCLUDED
#define QUEUE_H_INCLUDED

#include <string>
#include <exception>

///Definicion de la Cola
template <class T>
class Queue {       ///LDLCCED
    private:
        ///Definicion del Nodo
        class Node {//Tendra un encabezado Dummy
            private:
                T* dataPtr;
                Node* next;
                Node* prev;

            public:
                ///Definicion de Excepcion para el Nodo
                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 T&);

                ~Node();

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

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

```
    public:
        typedef Node* Position;

        ///Definicion de Excepcion para la Cola
        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();
                    }
            };

    private:
        Node* header;

        void copyAll(const Queue<T>&);//Guardara el objeto creado a una
copia
        void deleteAll();

    public:
        ///Constructores
        Queue();
        Queue(const Queue&);

        ~Queue(); ///Destructor

        ///Sobrecarga de operador
        Queue& operator = (const Queue&);//No es necesario el <T> porque
para el compilador ya esta todo esta plantillado asi que ya no es
necesario

        ///Metodos para una Cola
        bool isEmpty()const;

        void enqueue(const T&);

        T dequeue();

        T getFront()const;
    };

///Implementacion
using namespace std;

///del Nodo
template <class T>
Queue<T>::Node::Node():dataPtr(nullptr), next(nullptr), prev(nullptr) {}

template <class T>
Queue<T>::Node::Node(const T& e): dataPtr(new T(e)), next(nullptr),
prev(nullptr) {
```

```
    if(dataPtr == nullptr) {
        throw Exception ("Memoria insuficiente, creando nodo");
        }
    }

template <class T>
Queue<T>::Node::~Node() {
    delete dataPtr;
    }

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

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

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

template <class T>
typename Queue<T>::Position Queue<T>::Node::getPrev()const {
    return prev;
    }

template <class T>
void Queue<T>::Node::setDataPtr(T* p) {
    dataPtr = p;
    }

template <class T>
void Queue<T>::Node::setData(const T& e) {
    if(dataPtr == nullptr) {
        if((dataPtr = new T(e)) == nullptr) {
            throw Exception ("Memoria no disponible, setData");
            }
        }
    else {
        *dataPtr = e;
        }
    }
template <class T>
void Queue<T>::Node::setNext(Node* p) {
    next = p;
    }
template <class T>
void Queue<T>::Node::setPrev(Node* p) {
    prev = p;
    }
```

```
///de la Cola
///Metodos privados
template <class T>
void Queue<T>::copyAll(const Queue<T>& q) {
    Position aux(q.header->getNext());
    Position newNode;

    while(aux != q.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 T>
void Queue<T>::deleteAll() {
    Position aux;

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

        delete aux;
        }

    header->setPrev(header);
    }

///Constructores
template <class T>
Queue<T>::Queue(): header(new Node) {
    if(header == nullptr) {
        throw Exception("Memoria no disponible, inicializando Queue");
        }

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

template <class T>
Queue<T>::Queue(const Queue& q): Queue() {//Llama otra vez al contructor
    copyAll(q);
    }
```

```
template <class T>
Queue<T>::~Queue() {///Destructor
    deleteAll();
    delete header;
    }

///Sobre carga de operador
template <class T>
Queue<T>& Queue<T>::operator =(const Queue& q) {
    deleteAll();
    copyAll(q);

    return *this;
    }

///Metodos publicos de la Cola
template <class T>
bool Queue<T>::isEmpty() const {
    return header->getNext() == header;
    }

template <class T>
void Queue<T>::enqueue(const T& e) {
    Position aux;

    try {
        if((aux = new Node(e)) == nullptr) {
            throw Exception ("Memoria no disponible, enqueue");
            }
        }
    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 T>
T Queue<T>::dequeue() {
    if(isEmpty()) {
        throw Exception ("Insuficiencia de datos, dequeue");
        }

    T result (header->getNext()->getData());

    Position aux(header->getNext());

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

    delete aux;
    return result;
    }
```

```cpp
template <class T>
T Queue<T>::getFront() const {
    if(isEmpty()) {
        throw Exception ("Insuficiencia de datos, getFront");
        }

    return header->getNext()->getData();
    }

#endif // QUEUE_H_INCLUDED
```

"E:\Documentos PC\UDG Materias\ESTRUCTURA\A11 - Pila y Cola\Stack - Queue DINAMICA\bin\Debug\Stack - Queue.exe"

```
        Convertidor de Notacion de Infija a Posfija

Ingrese expresion con notacion infija: A/B^(C+D)-E*F/G^H
Conversion a notacion posfija: ABCD+^/EF*GH ^/-

Desea convertir otra expresion [S]i/[N]o): s

Ingrese expresion con notacion infija: (A-B)+(C/(D-E^F))/G*H
Conversion a notacion posfija: AB-CDEF^-/G/H *+

Desea convertir otra expresion [S]i/[N]o): s

Ingrese expresion con notacion infija: ((((((A+B)*C)-D)^E)/F)+G)*H
Conversion a notacion posfija: AB+C*D-E^F/G+H *

Desea convertir otra expresion [S]i/[N]o): s

Ingrese expresion con notacion infija: (A*B+C)*D/((E+A)-(A^B+C))-E^A
Conversion a notacion posfija: AB*C+D*EA+AB^C+-/EA ^-

Desea convertir otra expresion [S]i/[N]o): s

Ingrese expresion con notacion infija: (B*D)-((A+E)^C-E+A)/((D*A-C)/(E-A*B))
Conversion a notacion posfija: BD*AE+C^E-A+DA*C-EAB*-/ /-

Desea convertir otra expresion [S]i/[N]o): s

Ingrese expresion con notacion infija: ((A*(E-F+G))/C+B)*(E/T^B+U)*E
Conversion a notacion posfija: AEF-G+*C/B+ETB^/U+*E *

Desea convertir otra expresion [S]i/[N]o): s

Ingrese expresion con notacion infija: ((B/C+A)*((A*B)^(A+B*(E-B))^B+(C+A*B)*E))*A
Conversion a notacion posfija: BC/A+AB*ABEB-*+^B^CAB*+E*+*A *

Desea convertir otra expresion [S]i/[N]o): s

Ingrese expresion con notacion infija: (B-(A/B*(C^E))-B*D)^(((A*D^B)/(C-E*Q)+E*Q-A*C)^D-C*A)*(C*A/D^B)
Conversion a notacion posfija: BAB/CE^*-BD*-ADB^*CEQ*-/EQ*+AC*-D^CA*-^CA*DB^/ *

Desea convertir otra expresion [S]i/[N]o): N


Process returned 0 (0x0)   execution time : 4.379 s
Press any key to continue.
```

Conclusión:

En esta práctica fue fácil de realizar porque la teoría de la pila y de la cola es la misma que los temas anteriores, pero ahora aplicamos los conocimientos de los nodos y los apuntadores, en donde decidiremos si lo queremos hacer de forma lineal o circular pero lo interesante fue que el código principal se mantuvo igual debido el encapsulamiento de POO. Así que solo se modificó el funcionamiento de la cola y de la pila para la actividad y pude notar por los *tiempos de ejecución del programa* que en esta actividad *fue más rápida que la anterior*. Pero no solo fue solo hacer el cambio, sino que también tuve que analizar el código y entenderle porque cada método como sus atributos tenían características especiales debido a que en uno usaba el encabezado Dummy, que me fue difícil de entender porque el puntero siguiente del encabezado era el primer nodo de la cola, haciendo que me confundiera un poco, pero con mucho estudio y análisis pude llegar a entender los temas, ya que se hace de forma dinámica el almacenamiento de los datos y se más eficiente el manejo de los datos.