

5-4-2019

LA PILA Y LA COLA, IMPLEMENTACIÓN DINÁMICA



CUCEI

david gutierrez alvarez
Estructura de datos I

RESUMEN PERSONAL Y FORMA DE ABORDAR EL PROBLEMA

Como ya se sabe, la implementación de herencia se utilizó para resolver este ejercicio, por dificultades de tiempo no pude implementar algo que fuera eficiente, pero lo hice funcional

Main.cpp

```
#include <iostream>

#include "menu.h"

using namespace std;

int main() {

    Menu menu;

    cout << "Fin!" << endl;

    return 0;

}
```

Menu.h

```
#ifndef MENU_H
#define MENU_H

#include "stack.h"
#include "stack.cpp"
#include "queue.h"
#include "queue.cpp"
#include <iostream>

class Menu {
public:

    Menu() ;

    void converter(const std::string &);

    bool operatorValid(const char &);

    int precedencia(const char &);

};

#endif // MENU_H
```

Menu.cpp

```
#include "menu.h"
```

```

#include <string.h>

using namespace std;

Menu::Menu() {
    string continue_, operation;
    do{
        cout << "\t\t\t\t\t.:MENU:." << endl << endl
            << "Introduce una operacion infija: ";
        getline(cin, operation);
        converter(operation); /*convierte operacion infija a posfija*/
        cout << endl << "Desea introducir otra operacion: S/N" << endl;
        getline(cin, continue_);
        cout << endl << endl;
    } while(continue_ == "S" or continue_ == "s");
}

void Menu::converter(const string &infija) {
    //    int count = 0; ///con esto vere el inicio y fin de los parentesis
    Stack<char> pila;
    Queue<char> cola;

    for (size_t i = 0; i < infija.size(); i++) {
        cola.enqueue(infija.c_str()[i]); /*mete todos los datos en la cola*/
    }

    while (!cola.empty()) {
        /*mientras haya algun dato*/
        if(operatorValid(cola.getFront())) {
            /*si es un operador*/
            if(precedencia(cola.getFront()) == 4){
                /*insertar en pila*/
                pila.push(cola.getFront());
            }

```

```

        if(precedencia(cola.getFront()) == 5) {
            while (!pila.isEmpty() and pila.getTop() != '(') {
                /*extraer elemento de la pila y mostrarlo*/
                cout << pila.pop();
            }
            if(pila.getTop() == '(') {
                /*sacarlo de la pila pero sin mostrarlo*/
                pila.pop();
            }
        }

        if(precedencia(cola.getFront()) < 4) {
            /*si es un operador*/

            while(!pila.isEmpty() and precedencia(pila.getTop()) >=
precedencia(cola.getFront()) and precedencia(pila.getTop() != 4)) {
                /*mientras que la pila no este vacia y su tope tenga una precedencia
mayor*/

                /*sacar el untilo elemento y mostrarlo*/
                cout << pila.pop();
            }
            pila.push(cola.getFront());
        }

    } else {
        cout << cola.getFront();
    }

    cola.dequeue();
}

while (!pila.isEmpty()) {
    cout << pila.pop();
}
}

bool Menu::operatorValid(const char &data) {
    char operators[8] = "+-*/^()";

```

```

    for (size_t i = 0; i < 7; i++) {
        if(operators[i] == data) {
            return true;
        }
    }
    return false;
}

int Menu::precedencia(const char &operator_) {
    switch (operator_) {
        case '+':
        case '-': return 1;

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

        case '^': return 3;

        case '(': return 4;

        case ')': return 5;
    }
    return 0;
}

```

List.h

```

#ifndef LIST_H
#define LIST_H

#include <iostream>

template<typename Type>
class List {
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(); }

};

```

```

class Node {
private:
    Type data;
    Node *next;

public:
    Node();
    Node(const Type &);

    Type &getData();
    Node *getNext() const;

    void setData(const Type &);
    void setNext(Node *);

};

```

```

private:
    Node *anchor;

    bool validPos(Node*) const;
    void copyAll(const List &);

```

```

public:
    List();

```

```

List(const List &);

~List();

bool empty() const;

void insert(Node *, const Type &);
void erase(Node *);

Node *getFirst() const;
Node *getLast() const;
Node *getPrev(Node *) const;
Node *getNext(Node *) const;

Node *find(const Type &) const;
Type &retrieve(Node *);

std::string toString() const;

void deleteAll();

List &operator = (const List &);
};

/// Implementacion

/// Node ///

template<typename Type>
List<Type>::Node::Node() : next(nullptr) { }

template<typename Type>
List<Type>::Node::Node(const Type &e) : data(e), next(nullptr) { }

template<typename Type>
Type &List<Type>::Node::getData() {
    return data;
}

```



```

template<typename Type>
typename List<Type>::Node* List<Type>::Node::getNext() const {
    return next;
}

```

```

template<typename Type>
void List<Type>::Node::setData(const Type &e) {
    data = e;
}

```

```

template<typename Type>
void List<Type>::Node::setNext(List<Type>::Node *p) {
    next = p;
}

```

```

/// List ///

```

```

template<typename Type>
bool List<Type>::validPos(List<Type>::Node *p) const {
    if(empty()) {
        return false;
    }

    Node * aux(anchor);

    do {
        if(aux == p) {
            return true;
        }
        aux = aux->getNext();
    }while (aux != anchor);

    return false;
}

```

```

template<typename Type>

void List<Type>::copyAll(const List &l) {

    Node *aux(l.anchor);

    Node *last(nullptr);

    Node *newNode;

    do{

        newNode = new Node(aux->getData());

        if(newNode == nullptr) {

            throw List<Type>::Exception("Memoria no disponible, coplyAll");

        }

        if(last == nullptr) {

            anchor = newNode;

        } else {

            last->setNext(newNode);

        }

        last = newNode;

        aux = aux->getNext();

    } while (aux != l.anchor);

    last->setNext(anchor);

}

template<typename Type>

List<Type>::List() : anchor(nullptr) { }

template<typename Type>

List<Type>::List(const List &l) {

    copyAll(l);

}

template<typename Type>

List<Type>::~~List() {

    deleteAll();

}

```

```
}
```

```
template<typename Type>
```

```
List<Type> &List<Type>::operator = (const List<Type> &l) {
```

```
    deleteAll();
```

```
    copyAll(l);
```

```
    return *this;
```

```
}
```

```
template<typename Type>
```

```
bool List<Type>::empty() const {
```

```
    return anchor == nullptr;
```

```
}
```

```
template<typename Type>
```

```
void List<Type>::insert(List<Type>::Node *p, const Type &e) {
```

```
    if(p != nullptr and !validPos(p)) {
```

```
        throw Exception("posicion invalida, insert");
```

```
    }
```

```
    Node *aux(new Node(e));
```

```
    if(aux == nullptr) {
```

```
        throw Exception("memoria no disponible, insert");
```

```
    }
```

```
    if(p == nullptr) { // inserta al principio
```

```
        if(empty()) { // insertar el primer elemento
```

```
            aux->setNext(aux);
```

```
        } else { // no es el primer elemeneto
```

```
            aux->setNext(anchor);
```

```
            getLast()->setNext(aux);///
```

```
        }
```

```
    anchor = aux;
```

```

    } else { // insertar en otra posicion
        aux->setNext(p->getNext());
    }
}

template<typename Type>
void List<Type>::erase(List<Type>::Node *p) {
    if(!validPos(p)) {
        throw Exception("posicion invalida, erase");
    }

    if(p == anchor) { // eliminar el primero
        if(p->getData() == p) { // es nodo unico
            anchor = nullptr;
        } else { // todavia hay mas de un nodo
            getLast()->setNext(p->getNext());
            anchor = anchor->getNext();
        }
    } else { // eliminar otro
        getPrev(p)->setNext(p->getNext());
    }
    delete p;
}

template<typename Type>
typename List<Type>::Node *List<Type>::getFirst() const {
    return anchor;
}

template<typename Type>
typename List<Type>::Node *List<Type>::getLast() const {
    if(empty()) {
        return nullptr;
    }

    Node *aux(anchor);

    while (aux->getNext() != nullptr) {

```

```

        aux = aux->getNext();

    }

    return aux;
}

template<typename Type>
typename List<Type>::Node *List<Type>::getPrev(List<Type>::Node *p) const {
    if(p == anchor){
        return nullptr;
    }

    Node *aux(anchor);

    do {
        if(aux->getNext() == p) {
            return aux;
        }

        aux = aux->getNext();
    } while (aux != anchor);

    return nullptr;
}

template<typename Type>
typename List<Type>::Node *List<Type>::getNext(List<Type>::Node *p) const {
    if(!validPos(p) or p->getNext() == anchor) { // encapsulamiento
        return nullptr;
    }

    return p->getNext();
}

template<typename Type>
typename List<Type>::Node *List<Type>::find(const Type &e) const {
    Node *aux(anchor);

    while (aux != nullptr and aux->getData() != e) {

```

```

        aux = aux->getNext();

    }

    return aux;
}

template<typename Type>
Type &List<Type>::retrieve(List<Type>::Node *p) {
    if(!validPos(p)) {
        throw Exception("posicion invalida, retrieve");
    }

    return p->getData();
}

template<typename Type>
std::string List<Type>::toString() const {
    std::string result;

    if(!empty()) {
        Node * aux(anchor);

        do {
            result += aux->getData().toString() + "\n";
            aux = aux->getNext();
        } while (aux != nullptr);
    }

    return result;
}

template<typename Type>
void List<Type>::deleteAll() {
    if(empty()) {
        return;
    }

    Node *mark(anchor);
    Node *aux;

```

```

do {

    aux = anchor;

    anchor = anchor->getNext();

    delete aux;

} while (anchor != nullptr);

}

#endif // LIST_H

```

Queue.h

```

#ifndef QUEUE_H
#define QUEUE_H

#include <stdexcept>
#include <string>
#include <iostream>
#include "list.h"

template <class Type>
class Queue : public List<Type>{
private:
    Type data;
public:
    Queue();
    bool empty();
    bool full();
    void enqueue(const Type &);
    Type dequeue();
    Type getFront();
};

#endif // QUEUE_H

```

Queue.cpp

```

#include "queue.h"

```

```

using namespace std;

template<class Type>
Queue<Type>::Queue() : List<Type>(nullptr) { }

template<class Type>
bool Queue<Type>::empty() {
    return List<Type>::empty();
}

template<class Type>
void Queue<Type>::enqueue(const Type &e) {
    this->insert(this->getLast(), e);
}

template<class Type>
Type Queue<Type>::dequeue() {
    if(empty()) {
        throw invalid_argument("insuficiencia de datos, dequeue");
    }
    this->erase(this->getFirst());
}

template<class Type>
Type Queue<Type>::getFront() {
    if(empty()) {
        throw invalid_argument("insuficiencia de datos, getFront");
    }
    return this->getFirst()->getData();
}

```

stack.h

```

#include "stack.h"
#include <stdexcept>

```



```

using namespace std;

template<class Type>
Stack<Type>::Stack() : List<Type>(nullptr) { }

template<class Type>
bool Stack<Type>::isEmpty() {
    return List<Type>::empty();
}

template<class Type>
void Stack<Type>::push(const Type &newData) {
    this->insert(this->getFirst(), newData);
}

template<class Type>
Type Stack<Type>::pop() {
    if(isEmpty()) {
        throw invalid_argument("insuficiencia de datos");
    }
    this->erase(this->getFirst());
}

template<class Type>
Type Stack<Type>::getTop() {
    if(isEmpty()) {
        throw invalid_argument("insuficiencia de datos");
    }
    return this->getFirst()->getData();
}

```

stack.h

```
#ifndef STACK_H
```

```
#define STACK_H

#include <exception>
#include <string>
#include "list.h"

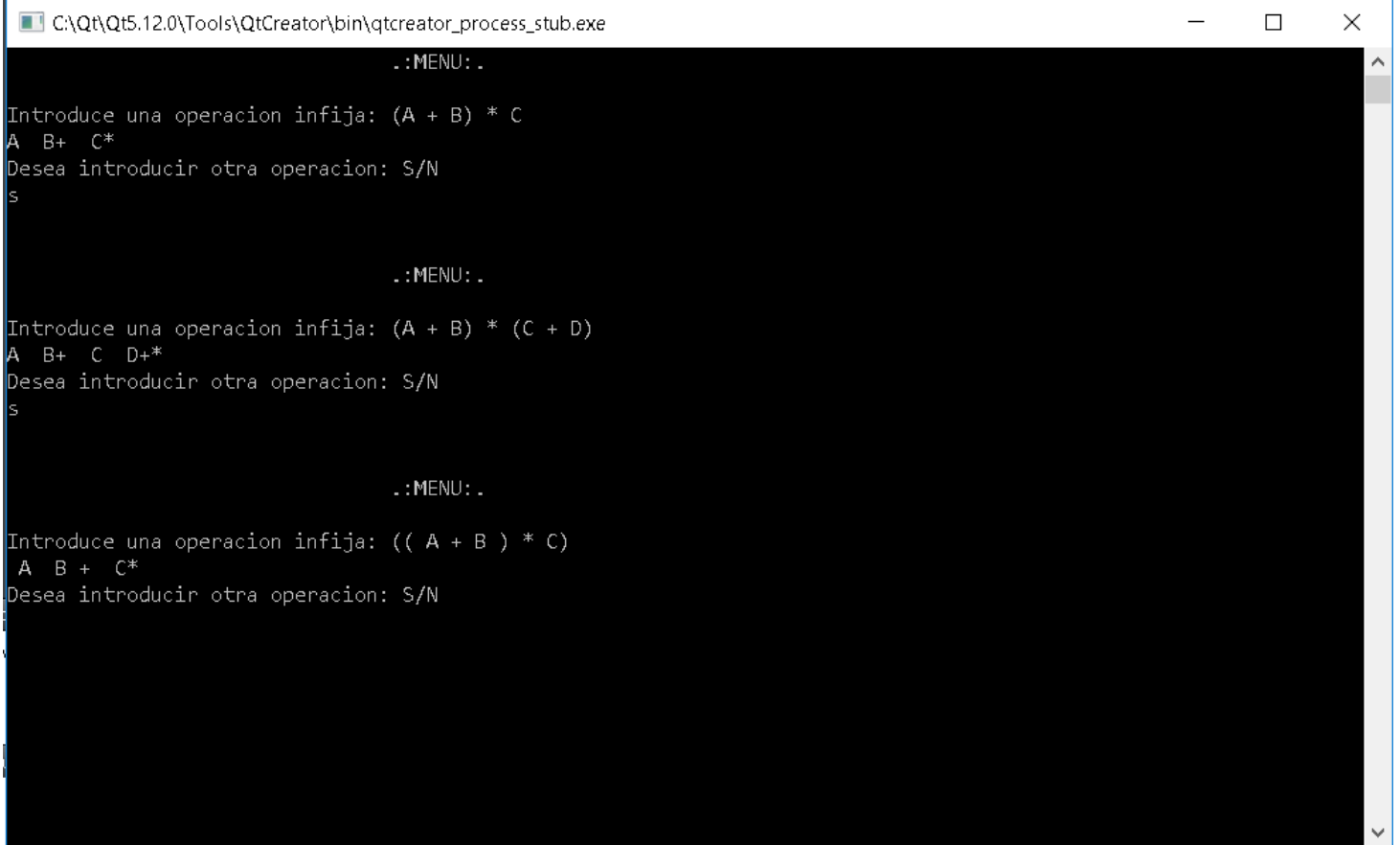
template <class Type>
class Stack : public List<Type>{
private:
    Type data;
public:
    Stack() ;

    bool isEmpty() ;
    //    bool isFull() ;

    void push(const Type&) ;
    Type pop() ;
    Type getTop() ;
};

#endif // STACK_H
```

CAPTURAS DE PANTALLA



```
C:\Qt\Qt5.12.0\Tools\QtCreator\bin\qtcreator_process_stub.exe

      .:MENU:..

Introduce una operacion infija: (A + B) * C
A B+ C*
Desea introducir otra operacion: S/N
s

      .:MENU:..

Introduce una operacion infija: (A + B) * (C + D)
A B+ C D+*
Desea introducir otra operacion: S/N
s

      .:MENU:..

Introduce una operacion infija: (( A + B ) * C)
A B + C*
Desea introducir otra operacion: S/N
s
```

Funciona correctamente :3