1-2-2019

APLICACIÓN DE PILA Y COLA

CUCEI

david gutierrez alvarez

Estructura de datos I

RESUMEN PERSONAL Y FORMA DE ABORDAR EL PROBLEMA

Esta actividad no tuvo complicaciones, cada vez siento que aprendo más, pero también me doy cuenta de que lo que se es muy poco a lo que debo saber, gracias a esto me he dado cuenta de que tan profundo puedo entrar en los temas sabiendo que el tiempo que a transcurrido a sido muy poco.

Para empezar a hacer esta actividad fue hacer un bloque de pseudocodigo, con esto fui viendo los posibles errores que podrían salir, al corregir estos, pase el código a c++ y me puse a trabajar en lo que se pide, no fue tan complicado como esperaba, la única complicación que tuve fue un errores de dedo al retornar un dato, pero fuera de esto fue una actividad un tanto sencilla.

Main.cpp

```cpp
#include <iostream>

#include "menu.h"
using namespace std;

int main() {
    Menu menu;
    cout << "Fin!" << endl;
    return 0;
}
```

Menu.h

```cpp
#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

```cpp
#include "menu.h"
#include <string.h>

using namespace std;



Menu::Menu() {
    string continue_, operation;
    do{
        cout << "\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()) {
        /*mentras 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;
}
```

Queue.h

```
#ifndef QUEUE_H
#define QUEUE_H
#include <stdexcept>
#include <string>
#include <iostream>

template <class Type, int ARRAYSIZE = 512>
class Queue {
private:
    Type data[ARRAYSIZE];
    int frontPos;
    int endPos;

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, int ARRAYSIZE>
Queue<Type, ARRAYSIZE>::Queue() : frontPos(0), endPos(ARRAYSIZE-1) { }

template<class Type, int ARRAYSIZE>
bool Queue<Type, ARRAYSIZE>::empty() {
    return (frontPos == endPos+1) or
           ((frontPos == 0) and (endPos == ARRAYSIZE-1));
}

template<class Type, int ARRAYSIZE>
bool Queue<Type, ARRAYSIZE>::full() {
    return (frontPos == endPos+2) or
           ((frontPos == 0) and (endPos == ARRAYSIZE-2)) or
           ((frontPos == 1) and (endPos == ARRAYSIZE-1));
}

template<class Type, int ARRAYSIZE>
void Queue<Type, ARRAYSIZE>::enqueue(const Type &e) {
    if(full()) {
        throw invalid_argument("desbordamiento de datos");
```

```
    }
//    endPos++;
//    endPos = (++endPos == ARRAYSIZE) ? 0 : endPos;
    data[ endPos = (++endPos == ARRAYSIZE) ? 0 : endPos] = e;
}

template<class Type, int ARRAYSIZE>
Type Queue<Type, ARRAYSIZE>::dequeue() {
    if(empty()) {
        throw invalid_argument("insuficiencia de datos, dequeue");
    }
    Type result(data[frontPos]);

    frontPos = (++frontPos == ARRAYSIZE ? 0 : frontPos);

    return result;

}

template<class Type, int ARRAYSIZE>
Type Queue<Type, ARRAYSIZE>::getFront() {
    if(empty()) {
        throw invalid_argument("insuficiencia de datos, getFront");
    }
    return data[frontPos];
}
```

stack.h

```
#ifndef STACK_H
#define STACK_H

#include <exception>
#include <string>


class StackException : public std::exception {
private:
    std::string asg;
public:
//    explicit exception();
};


template <class T, int ARRAYSIZE = 512>
class Stack {
private:
    T data[ARRAYSIZE];
    int top;

public:
    Stack();

    bool isEmpty();
    bool isFull();

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

    T getTop();

 };

#endif // STACK_H
```

stack.h

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

using namespace std;

template<class T, int ARRAYSIZE>
Stack<T, ARRAYSIZE>::Stack() : top(-1) { }


template<class T, int ARRAYSIZE>
bool Stack<T, ARRAYSIZE>::isEmpty() {
    return top == -1;
}

template<class T, int ARRAYSIZE>
bool Stack<T, ARRAYSIZE>::isFull() {
    return top == ARRAYSIZE-1;
}

template<class T, int ARRAYSIZE>
void Stack<T, ARRAYSIZE>::push(const T &newData) {
    if(isFull()) {
        throw invalid_argument("desbordamiento de datos");
    }
    data[++top] = newData;
}

template<class T, int ARRAYSIZE>
T Stack<T, ARRAYSIZE>::pop() {
    if(isEmpty()) {
        throw  invalid_argument("insuficiencia de datos");
    }
    return data[top--];
}

template<class T, int ARRAYSIZE>
T Stack<T, ARRAYSIZE>::getTop() {
    if(isEmpty()) {
        throw invalid_argument("insuficiencia de datos");
    }
    return data[top];
}
```

CAPTURAS DE PANTALLA