**Main.cpp**

#include <iostream>

#include <fstream>

#include <sstream>

#include "StackOnList.h"

int max(int a,int b) {

return a>b?a:b;

}

int min(int a,int b) {

return a<b?a:b;

}

***//Задание 4. Использование стека для решения задач***

***//Вычислить значение формулы с min, max***

***//Пусть дана без ошибок формула, имеющая следующий синтаксис:***

***//<формула>::=<цифра> | max(<формула>, < формула >) | min (<формула>,< формула >)***

***//<цифра>::=0 1 2 3 4 5 6 7 8 9***

***//Например, 8 или max(4, min(3, 5)) или min(min(3, 5), max(2, max(3, 4))).***

***//Вычислить значение формулы, используя структуру данных стек.***

int Result(std::string);

int main() {

StackOnList<int> first;

first.Push(5);

std::cout<<first.Size()<<std::endl;//1

std::cout<<first.Top()->value<<std::endl;//5

std::cout<<first.Pop()<<std::endl;//5

std::cout<<first.Size()<<std::endl;//0

std::cout<<first.IsEmpty()<<std::endl;//1

first.Push(6);

StackOnList<int> second=std::move(first);

std::cout<<second.Pop()<<std::endl;//6

std::cout<<second.Size()<<std::endl;//0

second.Push(7);

StackOnList<int> third(second);

std::cout<<(third==second)<<std::endl;//1

std::cout<<(third==first)<<std::endl;//0

int value=third.Pop();

StackOnList<int> fourth=third;//Assertion failed! Expression: stackOnList.size!=0 && "list is empty"

//Задание4

int result=Result("C:\\Clion projects\\practice\\practice\_8\_StackOnList\\input.txt");

std::cout<<result;//3

return 0;

}

int Result(std::string file\_name){

StackOnList<char> Formula;

std::ifstream fin(file\_name);

if (!fin.is\_open()){

std::cout<<"Error file open";

}

int result;

std::string input;

std::getline(fin,input);

std::stringstream input\_str(input);

while(!input\_str.eof()){

char element;

input\_str>>element;

if(element=='m'){

input\_str>>element;

Formula.Push(element);

} else if (isdigit(element)){

Formula.Push(element);

}

else if(element==')'){

char element\_1=Formula.Pop();

char element\_2=Formula.Pop();

char operation=Formula.Pop();

if(operation=='a'){

result=max(atoi(&element\_1),atoi(&element\_2));

Formula.Push((char)result);

}

else if(operation=='i'){

result=min(atoi(&element\_1),atoi(&element\_2));

Formula.Push((char)result);

}

}

}

return Formula.Pop();

}

**NodeStack.h**

#ifndef PRACTICE\_8\_STACKONLIST\_NODESTACK\_H

#define PRACTICE\_8\_STACKONLIST\_NODESTACK\_H

***Структура NodeStack должна содержать:***

***• o Публичное поле value типа int со значением, хранимым в данном элементе списка.***

***• o Приватные поля для осуществления связи элементов в списке.***

***• o Приватный конструктор (ничто кроме StackOnList не должно иметь возможности создания элементов типа Node).***

template<typename T>

struct NodeStack{

public:

T value;

private:

NodeStack\* next;

NodeStack()=default;

NodeStack(T \_value, NodeStack\* \_next):value(\_value),next(\_next){};

template<typename Type>

friend class StackOnList;

};

#endif //PRACTICE\_8\_STACKONLIST\_NODESTACK\_H

**StackOnList.h**

#ifndef PRACTICE\_8\_STACKONLIST\_STACKONLIST\_H

#define PRACTICE\_8\_STACKONLIST\_STACKONLIST\_H

#include "assert.h"

#include <initializer\_list>

#include "NodeStack.h"

***/\* Задание 2.Создать класс для реализации Стека на последовательности связных компонент StackOnList и публичную структуру NodeStack.***

***Класс StackOnList должны иметь следующий публичный интерфейс:***

***• o Конструктор по умолчанию.***

***• o Конструктор от std::initializer\_list.***

***• o Методы Size() и IsEmpty().***

***• o Метод Push(const int& value), осуществляющий вставку элемента в вершину стека.***

***• o Метод NodeStack\* Top(), позволяющие получить первый элемент в стеке, не извлекая его.***

***• o Метод int Pop(), осуществляющий извлечение первого элемента из стека и возвращающий его значение.***

***• o Проверить корректность вызовов и передаваемых параметров с помощью assertов.***

***• o Создать константный аналог для метода Top, который будет возвращать const NodeStack\*.***

***• o Обеспечить возможность проверки на равенство/различие стеков при помощи соответствующих операторов.***

***• o Реализовать конструктор копирования и копирующий оператор присваивания.***

***• o Реализовать конструктор перемещения и перемещающий оператор присваивания.\*/***

***//Параметризовать класс из Задания 2.***

template<class T>

class StackOnList{

public:

StackOnList():size(0),top(nullptr){};

StackOnList(const StackOnList& stackOnList);

StackOnList(StackOnList&& stackOnList);

StackOnList(std::initializer\_list<T>& list);

int Size(){return size;};

bool IsEmpty(){return size==0;};

void Push(const int& value);

NodeStack<T>\* Top(){return top;};

NodeStack<T>\* Top()const{return top;};

T Pop();

bool operator ==(const StackOnList& stackOnList);

bool operator !=(const StackOnList& stackOnList);

StackOnList& operator=(const StackOnList& stackOnList);

StackOnList& operator=( StackOnList&& stackOnList);

private:

int size;

NodeStack<T>\* top;

};

template<typename T>

StackOnList<T>::StackOnList(const StackOnList<T>& stackOnList):size(stackOnList.size){

assert(stackOnList.size!=0 && "list is empty");

/\* if(stackOnList.size==0){

StackOnList();

return;

}\*/

NodeStack<T>\* element=new NodeStack<T>(stackOnList.top->value, nullptr);

top=element;

NodeStack<T>\* cur\_element=stackOnList.top->next;

while (cur\_element!= nullptr){

element->next=new NodeStack<T>(cur\_element->value, nullptr);

element=element->next;

cur\_element=cur\_element->next;

}

}

template <typename T>

StackOnList<T>::StackOnList(StackOnList&& stackOnList):top(stackOnList.top),size(stackOnList.size){

stackOnList.top= nullptr;

stackOnList.size=0;

}

template <typename T>

StackOnList<T>& StackOnList<T>:: operator=(const StackOnList& stackOnList){

//если объекты равны

if(this==&stackOnList){

return \*this;

}

//освобождение памяти объекта для записи новых данных из другого объекта

while(top!= nullptr){

NodeStack<T>\* temp=top;

top=top->next;

delete temp;

}

//если размер объекта, который мы копируем=0

size=stackOnList.size;

if(size==0){

top= nullptr;

return \*this;

}

//копирование элементов

NodeStack<T>\* element=new NodeStack<T>(stackOnList.top->value, nullptr);

top=element;

NodeStack<T>\* cur\_element=stackOnList.top->next;

while (cur\_element!= nullptr){

element->next=new NodeStack<T>(cur\_element->value, nullptr);

element=element->next;

cur\_element=cur\_element->next;

}

}

template<typename T>

StackOnList<T>& StackOnList<T>:: operator=( StackOnList&& stackOnList) {

//освобождение памяти объекта для записи новых данных из другого объекта

while (top != nullptr) {

NodeStack<T> \*temp = top;

top = top->next;

delete temp;

}

top=stackOnList.top;

size=stackOnList.size;

stackOnList.top= nullptr;

stackOnList.size=0;

}

template <typename T>

StackOnList<T>::StackOnList(std::initializer\_list<T>& list):StackOnList(){

for(T value:list){

Push(value);

}

}

template <typename T>

void StackOnList<T>::Push(const int& value){

NodeStack<T>\* new\_element=new NodeStack<T>(value, top);

top=new\_element;

size++;

}

template <typename T>

T StackOnList<T>::Pop(){

assert(size!=0 && "Stack is empty");

NodeStack<T>\* top\_element=top;

top=top->next;

T value=top\_element->value;

delete top\_element;

--size;

return value;

}

template<typename T>

bool StackOnList<T>::operator==(const StackOnList<T> &stackOnList) {

if(size!=stackOnList.size){

return false;

}

NodeStack<T> \*elem1=top;

NodeStack<T> \*elem2=stackOnList.top;

while(elem1!= nullptr){

if(elem1->value!=elem2->value){

return false;

}

elem1=elem1->next;

elem2=elem2->next;

}

return true;

}

template<typename T>

bool StackOnList<T>::operator!=(const StackOnList<T> &stackOnList) {

return !(this==stackOnList);

}

#endif //PRACTICE\_8\_STACKONLIST\_STACKONLIST\_H