**МИНОБРНАУКИ РОССИИ**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ  
ВЫСШЕГО ОБРАЗОВАНИЯ  
**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ  
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В.Г.ШУХОВА»  
(БГТУ им. В.Г.Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа №8

Дисциплина: Объектно-ориентированное программирование

по теме Создание шаблонов классов в С++.

Выполнил: ст. группы ВТ-22  
Макаров Даниил Сергеевич

Проверил: Буханов Д.Г.

**Белгород 2019**

**Лабораторная работа № 9**

**Использование стандартной библиотеки шаблонов STL**

**Цель работы:** знакомство со стандартной библиотекой шаблонов в С++; получение навыков использования классов контейнеров, итераторов, алгоритмов.

**Содержание работы**

Разработать программное обеспечения для решения соответствующего варианта. Оформить отчет. Для реализации поставленных задач требуется использовать следующие библиотеки классов: list, vector, queue, iostream, algorithm, set, iterator, map, stack.

**Вариант 5.**

Разработать программное обеспечение для решения следующей задачи: преобразование математического выражения в обратную польскую запись, для хранения использовать stack, выполнить вычисления. Реализовать вычисление сложения, вычитания, и других арифметических операций над stack.

#ifndef RPN\_H

#define RPN\_H

#include <string>

#include <iostream>

#include <stack>

using namespace std;

const string STR\_PLUS = "+";

const string STR\_SUB = "-";

const string STR\_MULT = "\*";

const string STR\_DIV = "/";

const string STR\_OPEN\_BRC = "(";

const string STR\_CLOSE\_BRC = ")";

enum Token\_priority{

operand\_priority = 0,

bracket\_priority = 0,

mult\_priority = 2,

div\_priority = 2,

sum\_priority = 1,

sub\_priority = 1

};

enum Token\_id{

operand\_id = 0,

sum\_id = 1,

subst\_id = 2,

mult\_id = 3,

div\_id = 4,

open\_brc\_id = 5,

close\_brc\_id = 6,

};

class Token{

public:

virtual int calc(int a,int b) = 0;

virtual int get\_value() = 0;

virtual string get\_str() = 0;

virtual Token\_priority get\_prior() = 0;

virtual Token\_id get\_id() = 0;

};

class Operand: public Token{

private:

int value;

public:

int calc(int a, int b){return 0;};

Operand(int value):value(value){};

int get\_value(){return this->value;};

void set\_value(int val){this->value=val;};

string get\_str(){return to\_string(this->value);};

Token\_priority get\_prior(){return operand\_priority;};

Token\_id get\_id(){return operand\_id;};

};

class Open\_bracket: public Token{

public:

int calc(int a, int b){return 0;};

int get\_value(){return 0;};

string get\_str(){return STR\_OPEN\_BRC;};

Token\_priority get\_prior(){return bracket\_priority;};

Token\_id get\_id(){return open\_brc\_id;};

};

class Close\_bracket: public Token{

public:

int calc(int a, int b){return 0;};

int get\_value(){return 0;};

string get\_str(){return STR\_CLOSE\_BRC;};

Token\_priority get\_prior(){return bracket\_priority;};

Token\_id get\_id(){return close\_brc\_id;};

};

class OperatorPlus: public Token{

public:

int calc(int a,int b){

return a+b;

};

int get\_value(){return 0;};

Token\_priority get\_prior(){return sum\_priority;};

Token\_id get\_id(){return sum\_id;};

string get\_str(){return STR\_PLUS;};

};

class OperatorMinus: public Token{

public:

int calc(int a,int b){

return a-b;

};

int get\_value(){return 0;};

Token\_priority get\_prior(){return sub\_priority;};

Token\_id get\_id(){return subst\_id;};

string get\_str(){return STR\_SUB;};

};

class OperatorDiv: public Token{

public:

int calc(int a,int b){

return a/b;

};

int get\_value(){return 0;};

Token\_priority get\_prior(){return div\_priority;};

Token\_id get\_id(){return div\_id;};

string get\_str(){return STR\_DIV;};

};

class OperatorMult: public Token{

public:

int calc(int a,int b){

return a\*b;

};

int get\_value(){return 0;};

Token\_priority get\_prior(){return mult\_priority;};

Token\_id get\_id(){return mult\_id;};

string get\_str(){return STR\_MULT;};

};

Token\* create\_token(Token\_id id);

string infix\_to\_postfix(string str);

vector<Token\*> str\_to\_token(string str);

int calc\_tokens(vector<Token\*> tokens);

#endif

#include "rpn.hpp"

int calc\_tokens(vector<Token\*> tokens){

stack<int> calc\_stack;

int a,b;

for(int i=0;i<tokens.size();i++){

if(tokens[i]->get\_id()==operand\_id){

calc\_stack.push(tokens[i]->get\_value());

else{

a = calc\_stack.top();

calc\_stack.pop();

b = calc\_stack.top();

calc\_stack.pop();

calc\_stack.push(tokens[i]->calc(a,b));

}

};

};

string infix\_to\_postfix(string str){

string result;

string current\_token;

stack<Token\*> token\_stack;

vector<string> sep\_str;

//разбиение строки на список элементов

while(str.length()!=0){

int sep\_index=str.find(" ",0);

current\_token = str.substr(0,sep\_index);

str.erase(0,sep\_index+1);

sep\_str.push\_back(current\_token);

}

//непосредственно преобразование в ОПЗ

Token\* temp;

string temp\_str;

bool flag = true;

for(int i=0,size=sep\_str.size();i<size;i++){

if(sep\_str[i]==STR\_PLUS){

if(!token\_stack.empty()){

flag = true;

while(flag&&(token\_stack.top()->get\_prior()>sum\_priority)){

temp\_str=token\_stack.top()->get\_str();

result=result+temp\_str+" ";

token\_stack.pop();

if(token\_stack.empty()){

flag = false;

};

};

};

temp=create\_token(sum\_id);

token\_stack.push(temp);

}else if(sep\_str[i]==STR\_SUB){

if(!token\_stack.empty()){

flag = true;

while(flag&&(token\_stack.top()->get\_prior()>sub\_priority)){

temp\_str=token\_stack.top()->get\_str();

token\_stack.pop();

result=result+temp\_str+" ";

if(token\_stack.empty()){

flag = false;

};

};

};

temp=create\_token(subst\_id);

token\_stack.push(temp);

}else if(sep\_str[i]==STR\_MULT){

if(!token\_stack.empty()){

flag = true;

while(flag&&(token\_stack.top()->get\_prior()>mult\_priority)){

temp\_str=token\_stack.top()->get\_str();

token\_stack.pop();

result=result+temp\_str+" ";

if(token\_stack.empty()){

flag = false;

};

};

};

temp=create\_token(mult\_id);

token\_stack.push(temp);

}else if(sep\_str[i]==STR\_DIV){

if(!token\_stack.empty()){

flag = true;

while(flag&&(token\_stack.top()->get\_prior()>div\_priority)){

temp\_str=token\_stack.top()->get\_str();

token\_stack.pop();

result=result+temp\_str+" ";

if(token\_stack.empty()){

flag = false;

};

};

};

temp=create\_token(div\_id);

token\_stack.push(temp);

}else if(sep\_str[i]==STR\_OPEN\_BRC){

temp=create\_token(open\_brc\_id);

token\_stack.push(temp);

}else if(sep\_str[i]==STR\_CLOSE\_BRC){

while(token\_stack.top()->get\_id()!=open\_brc\_id){

if(token\_stack.empty()){

cout<<"Неверно поставлен разделитель или скобки не согласованны.";

exit(1);

};

temp\_str=token\_stack.top()->get\_str();

token\_stack.pop();

result=result+temp\_str+" ";

};

token\_stack.pop();

}else{

result=result+sep\_str[i]+" ";

}

}

while(!token\_stack.empty()){

temp\_str=token\_stack.top()->get\_str();

token\_stack.pop();

result=result+temp\_str+" ";

}

return result;

};

Token\* create\_token(Token\_id id){

Token\* temp;

switch(id){

case operand\_id:

temp = new Operand(0);

break;

case sum\_id:

temp = new OperatorPlus();

break;

case subst\_id:

temp = new OperatorMinus();

break;

case mult\_id:

temp = new OperatorMult();

break;

case div\_id:

temp = new OperatorDiv();

break;

case open\_brc\_id:

temp = new Open\_bracket();

break;

case close\_brc\_id:

temp = new Close\_bracket();

break;

}

return temp;

};

vector<Token\*> str\_to\_token(string str){

vector<Token\*> token\_vector;

vector<string> sep\_str;

//разбиение строки на список элементов

while(str.length()!=0){

int sep\_index=str.find(" ",0);

string current\_token = str.substr(0,sep\_index);

str.erase(0,sep\_index+1);

sep\_str.push\_back(current\_token);

}

for(int i=0,size=sep\_str.size();i<size;i++){

if(sep\_str[i]==STR\_PLUS){

token\_vector.push\_back(create\_token(sum\_id));

}

else if(sep\_str[i]==STR\_SUB){

token\_vector.push\_back(create\_token(subst\_id));

}

else if(sep\_str[i]==STR\_MULT){

token\_vector.push\_back(create\_token(mult\_id));

}

else if(sep\_str[i]==STR\_DIV){

token\_vector.push\_back(create\_token(div\_id));

}

else{

Operand\* temp = new Operand(stoi(sep\_str[i]));

token\_vector.push\_back(temp);

}

}

};

#include <string>

#include <iostream>

#include <stack>

#include "rpn.hpp"

using namespace std;

int main(){

string str,result\_str;

vector<Token\*> token\_list;

cout<<"Введите строку в инфиксной нотации - ";

getline(cin,str);

result\_str = infix\_to\_postfix(str);

cout<<result\_str<<endl;

token\_list=str\_to\_token(result\_str);

cout<<"Результат = "<<calc\_tokens(token\_list);

return 0;

}