**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра CАПР**

отчет

**по курсовой работе**

**по дисциплине «Алгоритмы и структуры данных»**

**Вариант 1**

|  |  |  |
| --- | --- | --- |
| Студент гр. 9302 |  | Квитко Д.В. |
| Преподаватель |  | Тутуева.А.В |

Санкт-Петербург

2020

**Постановка задачи**

Нужно создать простой калькулятор, который будет переводить выражение из обычной записи в обратную польскую нотацию. Также он должен вычислять его и при некорректном вводе выводить позицию ошибочного символа.

**Обоснование выбора используемых структур данных**

У нас есть 2 стека и 1 очередь и главный класс методы которого будут осуществлять

вычисление выражения, которые нужны для реализации задачи.

|  |  |
| --- | --- |
| stacknumber | Это стек, в который записываются числа для дальнейшей обработки и использования для вычисления выражения |
| queue | Эта очередь используется для хранения унарных функций в стеке для операторов |
| stackoper | Это стек для операций |

**Используемые Unit tests**

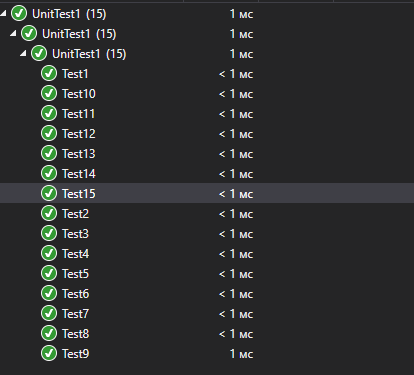
|  |  |
| --- | --- |
| Test1 | Проверка на сложение |
| Test2 | Проверка на разность |
| Test3 | Проверка на деление |
| Test4 | Проверка на умножение |
| Test5 | Проверка на возведение в степень |
| Test6 | Проверка на вычисление факториала |
| Test7 | Проверка на логарифм по основанию 10 |
| Test8 | Проверка на натуральный логарифм |
| Test9 | Проверка на косинус |
| Test10 | Проверка на синус |
| Test11 | Проверка на тангенс |
| Test12 | Проверка на котангенс |
| Test13 | Проверка на корень |
| Test14 | Проверка на перевод в обратную польскую нотацию |
| Test15 | Проверка на правильное решение выражений |

**Описание алгоритма решения**

Для того чтобы обычную запись перевести в обратную польскую нотация нужно записывать все операторы и числа, в стеки, которые были для них созданы. В случаи встречи числа при обработке выражения мы сразу же выводим это выражение в консоль. Когда мы работаем с операторами, то нам важно знать их приоритет, так как если приоритет нового оператора ниже или равен значениям операторов в стеке, то мы выводим эти все операторы, пока в начале стека не будет оператор, приоритет которого выше. Если приоритет оператора выше приоритета операторов в стеке, то они запишет его в стек. При выводе операторов происходит их применение для 2 или 1 числа в зависимости от оператора. При извлечении из стека операторов мы производим их действия между 2 последними числами, добавленными в стек, если это унарная функция, то с одним числом.

**Пример**





**Листинг**

Source.cpp

#include"reverse.h"

#include"stack.h"

#include<iostream>

#include <windows.h>

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

reverse\_Polish\_notation\* ship = new reverse\_Polish\_notation;

ship->call();

ship->invert();

std::cout << " = " << ship->result();

std::cout << std::endl << ship->postfix\_string;

return 0;

}

function.cpp

#include"reverse.h"

#include<iostream>

#include<string>

#include <cmath>

#include <sstream>

#define \_USE\_MATH\_DEFINES

reverse\_Polish\_notation::reverse\_Polish\_notation() {

infix\_string = "";

}

void reverse\_Polish\_notation::call() {

std::getline(std::cin, infix\_string);

}

reverse\_Polish\_notation::~reverse\_Polish\_notation() {

}

double fact(int N)

{

if (N < 0)

return 0;

if (N == 0)

return 1;

else

return N \* fact(N - 1);

}

void reverse\_Polish\_notation::invert() {

int size = infix\_string.length(), prioritet, prioret\_stack;;

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

if (infix\_string[i] == '0' || infix\_string[i] == '1'

|| infix\_string[i] == '2' || infix\_string[i] == '3'

|| infix\_string[i] == '4' || infix\_string[i] == '5'

|| infix\_string[i] == '6' || infix\_string[i] == '7'

|| infix\_string[i] == '8' || infix\_string[i] == '9' || infix\_string[i] == '.') {

postfix\_string += infix\_string[i];

std::cout << infix\_string[i];

if (infix\_string[i] == '.') {

number.head->integer = true;

}

if (now) {

now = false;

number.push(0);

}

number.head->numb += infix\_string[i];

number.head->date = atof(number.head->numb.c\_str());

}

else if (infix\_string[i] == 'p' && empty(i)) {

postfix\_string += infix\_string[i];

std::cout << infix\_string[i];

if (infix\_string[i + 1] == 'i') {

postfix\_string += infix\_string[i+1];

std::cout << infix\_string[i + 1];

number.push(acos(-1.0));

i++;

}

else

{

std::cout << "ошибка ввода";

return;

}

}

else if (infix\_string[i] == 'e' && empty(i)) {

postfix\_string += infix\_string[i];

std::cout << infix\_string[i];

number.push(exp(1));

}

else {

if (!now) {

number.head->index = i;

now = true;

skipspace = true;

postfix\_string += " ";

std::cout << " ";

}

prioritet = priority(infix\_string[i]);

if (prioritet == 1) {

operators.push(infix\_string[i]);

}

else if (prioritet == 0) {

for (; priority(operators.head->date) != 1; operators.pop\_front()) {

if (operators.head->date == '#') {

if (skipspace) {

std::cout << " ";

postfix\_string += " ";

}

skipspace = true;

realization(operators.head->date);

for (int j = 0; j < operators.head->Queue.size; operators.head->Queue.pop()) {

std::cout << operators.head->Queue.head->date;

postfix\_string += operators.head->Queue.head->date;

}

}

else {

if (skipspace) {

std::cout << " ";

postfix\_string += " ";

}

skipspace = true;

realization(operators.head->date);

std::cout << operators.head->date;

postfix\_string += operators.head->date;

}

std::cout << " ";

postfix\_string += " ";

}

operators.pop\_front();

}

else if (prioritet == 2 || prioritet == 3 || (prioritet == 4 && flag)) {

if (operators.head != nullptr) {

for (; operators.head != nullptr && priority(operators.head->date) != 1 && priority(operators.head->date) >= prioritet; operators.pop\_front()) {

if (skipspace) {

std::cout << " ";

postfix\_string += " ";

}

skipspace = true;

if (operators.head->date == '#') {

realization(operators.head->date);

for (int j = 0; j < operators.head->Queue.size; operators.head->Queue.pop()) {

std::cout << operators.head->Queue.head->date;

postfix\_string += operators.head->Queue.head->date;

}

}

else {

realization(operators.head->date);

std::cout << operators.head->date;

postfix\_string += operators.head->date;

}

std::cout << " ";

postfix\_string += " ";

}

}

operators.push(infix\_string[i]);

}

else if (prioritet == 4 && !flag) {

if (skipspace) {

std::cout << " ";

postfix\_string += " ";

}

skipspace = true;

flag = true;

operators.push('#');

if (infix\_string[i] == 'c' && empty(i)) {

if (empty(i + 1) && empty(i + 2)) {

if (infix\_string[i + 1] == 'o') {

if (infix\_string[i + 2] == 's') {

for (int j = i; j < i + 3; ++j)

operators.head->Queue.push(infix\_string[j]);

operators.head->index = i + 3;

i += 2;

}

else

{

std::cout << "Ошибка ввода ошибка в позиции: " << i + 3;

return;

}

}

else if (infix\_string[i + 1] == 't') {

if (infix\_string[i + 2] == 'g') {

for (int j = i; j < i + 3; ++j)

operators.head->Queue.push(infix\_string[j]);

operators.head->index = i + 3;

i += 2;

}

else

{

std::cout << "Ошибка ввода ошибка в позиции: " << i + 3;

return;

}

}

else

{

std::cout << "Ошибка ввода ошибка в позиции: " << i + 2;

return;

}

}

else

{

std::cout << "Ошибка ввода ошибка в позиции: " << i + 1;

return;

}

}

else if (infix\_string[i] == 'l') {

if (empty(i + 1)) {

if (infix\_string[i + 1] == 'o' && empty(i + 2)) {

if (infix\_string[i + 2] == 'g') {

for (int j = i; j < i + 3; ++j)

operators.head->Queue.push(infix\_string[j]);

operators.head->index = i + 3;

i += 2;

}

else

{

std::cout << "Ошибка ввода ошибка в позиции: " << i + 3;

return;

}

}

else if (empty(i + 1)) {

if (infix\_string[i + 1] == 'n') {

for (int j = i; j < i + 2; ++j)

operators.head->Queue.push(infix\_string[j]);

operators.head->index = i + 2;

i += 1;

}

else

{

std::cout << "Ошибка ввода ошибка в позиции: " << i + 2;

return;

}

}

else

{

std::cout << "Ошибка ввода ошибка в позиции: " << i + 2;

return;

}

}

else

{

std::cout << "Ошибка ввода ошибка в позиции: " << i + 1;

return;

}

}

else if (infix\_string[i] == 's') {

if (empty(i + 1) && empty(i + 2)) {

if (infix\_string[i + 1] == 'i') {

if (infix\_string[i + 2] == 'n') {

for (int j = i; j < i + 3; ++j)

operators.head->Queue.push(infix\_string[j]);

operators.head->index = i + 3;

i += 2;

}

else

{

std::cout << "Ошибка ввода ошибка в позиции: " << i + 3;

return;

}

}

else if (empty(i + 3)) {

if (infix\_string[i + 1] == 'q') {

if (infix\_string[i + 2] == 'r') {

if (infix\_string[i + 3] == 't') {

for (int j = i; j < i + 4; ++j)

operators.head->Queue.push(infix\_string[j]);

operators.head->index = i + 4;

i += 3;

}

else

{

std::cout << "Ошибка ввода ошибка в позиции: " << i + 4;

return;

}

}

else

{

std::cout << "Ошибка ввода ошибка в позиции: " << i + 3;

return;

}

}

else

{

std::cout << "Ошибка ввода ошибка в позиции: " << i + 2;

return;

}

}

else

{

std::cout << "Ошибка ввода ошибка в позиции: " << i + 2;

return;

}

}

else

{

std::cout << "Ошибка ввода ошибка в позиции: " << i + 1;

return;

}

}

else if (infix\_string[i] == 't') {

if (empty(i + 1)) {

if (infix\_string[i + 1] == 'g') {

for (int j = i; j < i + 2; ++j)

operators.head->Queue.push(infix\_string[j]);

operators.head->index = i + 2;

i += 1;

}

else

{

std::cout << "Ошибка ввода ошибка в позиции: " << i + 2;

}

}

else

{

std::cout << "Ошибка ввода ошибка в позиции: " << i + 1;

return;

}

}

else

{

std::cout << "Ошибка ввода ошибка в позиции: " << i + 1;

return;

}

}

}

}

if (operators.size != 0) {

for (; operators.head != nullptr; operators.pop\_front()) {

std::cout << " ";

postfix\_string += " ";

if (operators.head->date == '#') {

realization(operators.head->date);

for (int j = 0; j < operators.head->Queue.size; operators.head->Queue.pop()) {

postfix\_string += operators.head->Queue.head->date;

std::cout << operators.head->Queue.head->date;

}

}

else {

realization(operators.head->date);

postfix\_string += operators.head->date;

std::cout << operators.head->date;

}

}

}

rezult = number.head->date;

}

double reverse\_Polish\_notation::result() {

return rezult;

}

size\_t reverse\_Polish\_notation::priority(char symbol) {

switch (symbol)

{

case '^':

case '!':

case '#':

return 4;

case '\*':

case '/':

return 3;

case '-':

case '+':

return 2;

case '(':

return 1;

case ')':

return 0;

default:

flag = false;

return 4;

}

}

bool reverse\_Polish\_notation::empty(size\_t index) {

if (infix\_string.length() >= index + 1) {

return 1;

}

return 0;

}

void reverse\_Polish\_notation::realization(char operato) {

double shtoto;

switch (operato)

{

case '^': {

if (number.head == nullptr) {

std::cout << "недостаточно переменных на позиции: " << operators.head->index-1;

exit(0);

}

if (number.head->next == nullptr) {

std::cout << "недостаточно переменных на позиции: " << operators.head->index;

exit(0);

}

shtoto = pow(number.head->next->date, number.head->date);

number.pop\_front();

number.pop\_front();

number.push(shtoto);

break;

}

case '!':

{

if (number.head == nullptr) {

std::cout << "недостаточно переменных на позиции: " << operators.head->index;

exit(0);

}

shtoto = number.head->date;

if (number.head->integer) {

shtoto = sqrt(2 \* 3.1415 \* number.head->date) \* pow((number.head->date / 2.7182), number.head->date);

number.pop\_front();

number.push(shtoto);

}

else {

shtoto = fact(number.head->date);

number.pop\_front();

number.push(shtoto);

}

break;

}

case '\*':

{

if (number.head == nullptr) {

std::cout << "недостаточно переменных на позиции: " << operators.head->index-1;

exit(0);

}

if (number.head->next == nullptr) {

std::cout << "недостаточно переменных на позиции: " << operators.head->index;

exit(0);

}

shtoto = number.head->next->date \* number.head->date;

number.pop\_front();

number.pop\_front();

number.push(shtoto);

break;

}

case '/':

{

if (number.head == nullptr) {

std::cout << "недостаточно переменных на позиции: " << operators.head->index - 1;

exit(0);

}

if (number.head->next == nullptr) {

std::cout << "недостаточно переменных на позиции: " << operators.head->index;

exit(0);

}

shtoto = number.head->next->date / number.head->date;

number.pop\_front();

number.pop\_front();

number.push(shtoto);

break;

}

case '-':

{

if (number.head == nullptr) {

std::cout << "недостаточно переменных на позиции: " << operators.head->index - 1;

exit(0);

}

if (number.head->next == nullptr) {

std::cout << "недостаточно переменных на позиции: " << operators.head->index;

exit(0);

}

shtoto = number.head->next->date - number.head->date;

number.pop\_front();

number.pop\_front();

number.push(shtoto);

break;

}

case '+':

{

if (number.head == nullptr) {

std::cout << "недостаточно переменных на позиции: " << operators.head->index - 1;

exit(0);

}

if (number.head->next == nullptr) {

std::cout << "недостаточно переменных на позиции: " << operators.head->index;

exit(0);

}

shtoto = number.head->next->date + number.head->date;

number.pop\_front();

number.pop\_front();

number.push(shtoto);

break;

}

case '#':

{

if (number.head == nullptr) {

std::cout << "недостаточно переменных на позиции: " << operators.head->index;

exit(0);

}

if (operators.head->Queue.head->date == 'c') {

if (operators.head->Queue.head->next->date == 'o') {

shtoto = cos(number.head->date);

number.pop\_front();

number.push(shtoto);

}

else {

shtoto = 1 / tan(number.head->date);

number.pop\_front();

number.push(shtoto);

}

}

else if (operators.head->Queue.head->date == 'l') {

if (operators.head->Queue.head->next->date == 'o') {

shtoto = log10(number.head->date);

number.pop\_front();

number.push(shtoto);

}

else {

shtoto = log(number.head->date);

number.pop\_front();

number.push(shtoto);

}

}

else if (operators.head->Queue.head->date == 's') {

if (operators.head->Queue.head->next->date == 'i') {

shtoto = sin(number.head->date);

number.pop\_front();

number.push(shtoto);

}

else {

shtoto = sqrt(number.head->date);

number.pop\_front();

number.push(shtoto);

}

}

else if (operators.head->Queue.head->date == 't') {

shtoto = tan(number.head->date);

number.pop\_front();

number.push(shtoto);

}

}

default:

break;

}

}

queue.h

#ifndef queue\_H

#define queue\_H

class queue

{

private:

class queueNode {

public:

char date;

queueNode\* next;

queueNode(char date , queueNode\* next = nullptr)

{

this->date = date;

this->next = next;

};

~queueNode()

{

}

};

public:

queueNode\* head;

queueNode\* tail;

unsigned int size;

queue() {

tail = nullptr;

head = nullptr;

size = 0;

}

~queue() {

clear();

}

void clear() {

while (size != 0)

{

pop();

}

}

void push(char date) {

if (size == 0) {

head = new queueNode(date);

tail = head;

}

else {

tail->next = new queueNode(date);

tail = tail->next;

}

size++;

}

void pop() {

if (size == 0) {

return;

}

if (size == 1) {

delete head;

reset\_list();

}

else {

queueNode\* current = head;

head = head->next;

delete current;

}

size--;

}

void reset\_list()

{

head = nullptr;

tail = nullptr;

}

};

#endif

reverse.h

#ifndef reverse\_H

#define reverse\_H

#include<string>

#include"stack.h"

#include"stacknumber.h"

class reverse\_Polish\_notation

{

public:

std::string infix\_string;

std::string postfix\_string;

reverse\_Polish\_notation();

~reverse\_Polish\_notation();

void call();

void invert();

size\_t priority(char symbol);

bool empty(size\_t index);

void realization(char operators);

double result();

private:

stackoper operators;

stacknumber number;

double rezult;

bool flag = true, now = true, skipspace = false;

};

#endif

stack.h

#ifndef stack\_H

#define stack\_H

#include"queue.h"

class stackoper

{

private:

class stackNode {

public:

char date;

queue Queue;

stackNode\* next;

size\_t index;

stackNode(char date, stackNode\* next = nullptr)

{

this->date = date;

this->next = next;

};

~stackNode()

{

}

};

public:

stackNode\* head;

stackNode\* tail;

unsigned int size;

stackoper() {

head = nullptr;

tail = nullptr;

size = 0;

}

~stackoper() {

clear();

}

void clear() {

while (size != 0)

{

pop\_front();

}

}

void push(char date) {

if (size == 0) {

head = new stackNode(date);

tail = head;

}

else {

head = new stackNode(date, head);

}

size++;

}

void pushafter(char date) {

stackNode\* ggg = new stackNode(date);

ggg->next = head->next;

head->next = ggg;

size++;

}

void pop\_front() {

if (size == 0) {

return;

}

if (size == 1) {

delete head;

reset\_list();

}

else {

stackNode\* current = head;

head = head->next;

delete current;

}

size--;

}

void pop\_back() {

if (size == 0) return;

if (size == 1) {

delete head;

reset\_list();

}

else {

stackNode\* current = head;

while (current->next != tail) {

current = current->next;

}

current->next = nullptr;

delete tail;

tail = current;

}

size--;

}

void reset\_list()

{

head = nullptr;

tail = nullptr;

}

};

#endif

stacknumber.h

#ifndef stacknumber\_H

#define stacknumber\_H

#include"queue.h"

class stacknumber

{

private:

class stackNode {

public:

double date;

std::string numb;

stackNode\* next;

bool integer;

size\_t index;

stackNode(double date, stackNode\* next = nullptr)

{

this->date = date;

this->next = next;

};

~stackNode()

{

}

};

public:

stackNode\* head;

stackNode\* tail;

unsigned int size;

stacknumber() {

head = nullptr;

tail = nullptr;

size = 0;

}

~stacknumber() {

clear();

}

void clear() {

while (size != 0)

{

pop\_front();

}

}

void push(double date) {

if (size == 0) {

head = new stackNode(date);

tail = head;

}

else {

head = new stackNode(date, head);

}

size++;

}

void pushafter(char date) {

stackNode\* ggg = new stackNode(date);

ggg->next = head->next;

head->next = ggg;

size++;

}

void pop\_front() {

if (size == 0) {

return;

}

if (size == 1) {

delete head;

reset\_list();

}

else {

stackNode\* current = head;

head = head->next;

delete current;

}

size--;

}

void pop\_back() {

if (size == 0) return;

if (size == 1) {

delete head;

reset\_list();

}

else {

stackNode\* current = head;

while (current->next != tail) {

current = current->next;

}

current->next = nullptr;

delete tail;

tail = current;

}

size--;

}

void reset\_list()

{

head = nullptr;

tail = nullptr;

}

};

#endif