МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ  
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования   
**«Национальный исследовательский   
Нижегородский государственный университет им. Н.И. Лобачевского»**

**(ННГУ)**

**Институт информационных технологий, математики и механики**

Направление подготовки: «Программная инженерия»

**ОТЧЕТ**

по лабораторной работе

**Вычисление арифметических выражений (стеки)**

Выполнил:

студент группы 3822Б1ПР2

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Кондратьев Я.С.

Подпись

Нижний Новгород

2023 г.

Содержание

[1. Введение 3](#_Toc153920479)

[2. Постановка задачи 4](#_Toc153920480)

[3. Руководство пользователя 5](#_Toc153920481)

[4. Руководство программиста 6](#_Toc153920482)

[4.1. Описание структуры программы 6](#_Toc153920483)

[4.2. Описание структур данных 6](#_Toc153920484)

[4.2.1. TStack (шаблонный класс) 6](#_Toc153920485)

[4.2.2. TArithmeticExpression 6](#_Toc153920486)

[5. Эксперименты 7](#_Toc153920487)

[6. Заключение 8](#_Toc153920488)

[7. Литература 9](#_Toc153920489)

[8. Приложение 1 10](#_Toc153920490)

[9. Приложение 2 12](#_Toc153920491)

[10. Приложение 3 13](#_Toc153920492)

[11. Приложение 4 14](#_Toc153920493)

[12. Приложение 5 16](#_Toc153920494)

[13. Приложение 6 17](#_Toc153920495)

[14. Приложение 7 19](#_Toc153920496)

[15. Приложение 8 20](#_Toc153920497)

[16. Приложение 9 21](#_Toc153920498)

1. Введение

Цель лабораторной работы состоит в том, чтобы практически изучить динамическую структуру данных Стек. В ходе работы исследуются различные способы хранения стеков и разрабатываются методы и программы для решения задач, использующих стеки. Основной областью применения выбрано вычисление арифметических выражений, которое возникает при компиляции программ на высокоуровневом языке программирования в исполняемые программы.

При вычислении произвольных арифметических выражений возникают две ключевые задачи: проверка правильности введенного выражения и выполнение операций в определенном порядке с учетом приоритетов и расстановки скобок. Существует алгоритм, который позволяет вычислить произвольное арифметическое выражение за один просмотр без использования промежуточных результатов. Для применения этого алгоритма выражение должно быть представлено в постфиксной форме. Рассматриваемые в данной лабораторной работе алгоритмы являются введением в область машинных вычислений.

1. Постановка задачи

Арифметическое выражение - выражение, в котором операндами являются объекты, над которыми выполняются арифметические операции. Например, (1+2)/(3+4\*6.7)-5.3\*4.4

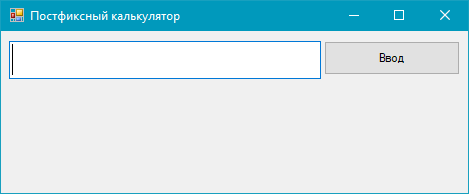
При такой форме записи (называемой инфиксной, где знаки операций стоят между операндами) порядок действий определяется расстановкой скобок и приоритетом операций. Постфиксная (или обратная польская) форма записи не содержит скобок, а знаки операций следуют после соответствующих операндов. Тогда для приведённого примера постфиксная форма будет иметь вид: 1 2+ 3 4 6.7\*+/ 5.3 4.4\* -

Обратная польская нотация была разработана австралийским ученым Чарльзом Хэмблином в середине 50-х годов прошлого столетия на основе польской нотации, которая была предложена в 1920 году польским математиком Яном Лукасевичем. Эта нотация лежит в основе организации вычислений для арифметических выражений. Известный ученый Эдсгер Дейкстра предложил алгоритм для перевода выражений из инфиксной в постфиксную форму. Данный алгоритм основан на использовании стека.

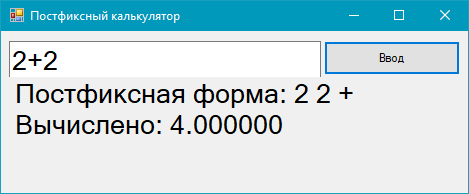
Стек (англ. stack), магазин – схема запоминания информации, при которой каждый вновь поступающий ее элемент как бы «проталкивает» вглубь отведенного участка памяти находящиеся там элементы (подобно патрону, помещаемому в магазин винтовки) и занимает крайнее положение (так называемую вершину стека). При выдаче информации из стека выдается элемент, расположенный в вершине стека, а оставшиеся элементы продвигаются к вершине; следовательно, элемент, поступивший последним, выдается первым [1]. Более строгое определение структуры дано в разделе 2 описания данной лабораторной работы.

1. Руководство пользователя

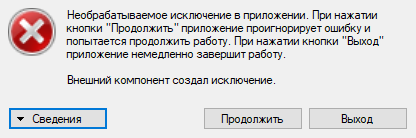
При запуске программы необходимо ввести два полинома в текстовые первого и второго блока соответствующе (также присутствуют полиномы по умолчанию). После чего выбрать операцию над введенными полиномами.

  
Рис. – Исходное состояние

После выбора операции будет выведен результат.

  
Рис. 2 – Результат

В случае возникновения ошибки во время исполнения, будет вызвано окно-сообщения с ошибкой

  
Рис. 3 – Сообщение об ошибке

1. Руководство программиста

## Описание структуры программы

Программа состоит из следующих модулей:

* Модуль Postfix, содержащий реализацию класса TArithmeticExpression (файлы в директориях ./include/ и ./src/).
* Модуль Postfix\_test. Набор тестов для класса TAritthmeticExpression. Включает в себя файл ./test/test\_expressuib.cpp. Реализованы они с помощью использования фреймворка Google Test.
* Примеры использования классов (файлы в директории ./samples/).
* Графическое приложение Postfix\_gui (файлы в директории ./gui/)

## Описание структур данных

### TStack (шаблонный класс)

protected:

size\_t size – размер стека

size\_t top – количество элементов

T\* mas – массив значений

public:

TStack(size\_t size = 1) – конструктор

TStack(TStack<T>& stack) noexcept – конструктор копирования

~TStack() noexcept - деструктор

void Push(T element) – положить элемент в стек

T Get() - получить индекс элемента

T TopView() – посмореть верхний элемент

size\_t GetSize() noexcept – получить размер стека

size\_t GetTop() noexcept – получить количество элементов

bool IsFull() noexcept – проверка на заполненность

bool IsEmpty() noexcept – проверка на опустошенность

TStack<T>& operator=(const TStack<T>& stack) noexcept;

friend std::ostream& operator<<(std::ostream& ostr, const TStack<T>& stack) noexcept – оператор вывода

friend std::istream& operator>>(std::istream& istr, const TStack<T>& stack) noexcept – оператор ввода

### TArithmeticExpression

std::string infix – поле инфиксная запись

std::string postfix – поле постфиксная запись

void ToPostfix() – приводит выражение к постфиксному виду, сохраняет результат в поле postfix  
 int GetPriority(char op) - возвращает приоритет оператора

TArithmeticExpression(std::string& infix) - конструктор

std::string GetInfix() const – получить инфиксную запись

std::string GetPostfix() const – получить постфиксную запись

double Calculate(const std::map < std::string , double > & values) – вычислить постфиксную запись

- перечисление констант для запуска тестов производительности

1. Эксперименты

Эксперименты проводились на ПК с следующими параметрами:

1. Операционная система: Windows 10
2. Процессор: Intel(R) Core(TM) i9-9900K CPU @ 3.60GHz 3.60 GHz
3. Версия Visual Studio: 2022

Эксперименты показали, что добавление и изъятие из очереди происходят за константое время, т.е. сложность алгоримов – .

1. Заключение

В итоге, данная практическая работа предоставляет возможность ознакомиться с использованием динамической структуры данных, известной как "Стек". В процессе выполнения лабораторной работы происходит исследование различных способов хранения стеков и разработка методов и программ для решения задач с использованием стеков. Основным фокусом работы является вычисление арифметических выражений, особенно в контексте компиляции программ, написанных на языке высокого уровня, в исполняемые программы.

Для решения задач вычисления произвольных арифметических выражений существует алгоритм, который позволяет вычислить выражение за один просмотр без необходимости сохранения промежуточных результатов. Для применения этого алгоритма выражение должно быть представлено в постфиксной форме. Разработанные алгоритмы, представленные в данной лабораторной работе, служат введением в область машинных вычислений.

Таким образом, данная лабораторная работа дает возможность практически освоить структуру данных Стек и получить начальные навыки в вычислении арифметических выражений в постфиксной форме. Эти навыки могут быть полезными при разработке программных решений в будущем.

1. Литература
   * + 1. Лабораторный практикум. Составители: Барышева И.В., Мееров И.Б., Сысоев А.В., Шестакова Н.В. Под редакцией Гергеля В.П. Учебно-методическое пособие. – Нижний Новгород: Нижегородский госуниверситет, 2017. – 105с.

1. Приложение 1

MyStack.h

#pragma once

#include <iostream>

template <class T>

class TStack {

protected:

int size;

int top;

T\* mas;

public:

TStack(int n = 0) {

if (n <= 0)

throw - 1;

size = n;// инициализация

top = 0;

mas = new T[size]();

}

TStack(TStack<T>& stack) {

size = stack.size;

top = stack.top;

mas = new T[size]();

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

mas[i] = stack.mas[i];

}

~TStack() {

if (mas != nullptr)

delete[] mas;

size = 0;

top = 0;

}

void Push(T a) {

if (IsFull()) {

throw - 1;

}

mas[top] = a;

top++;

}

T Get() {

if (IsEmpty()) {

throw - 1;

}

top--;

T tmp = mas[top];

mas[top] = T();

return tmp;

}

T TopView() { return mas[top - 1]; }

int GetSize()// индексы

{

return size;

}

int GetTop()

{

return top;

}

bool IsFull()

{

return top == size;

}

bool IsEmpty()

{

return top == 0;

}

TStack& operator=(TStack<T>& stack)

{

if (this == &stack)

return \*this;

if (mas != nullptr)

delete[] mas;

size = stack.size;

mas = new T[size];

top = stack.top;

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

mas[i] = stack.mas[i];

return \*this;

}

//операторы вводы и выводы

friend std::ostream& operator<<(std::ostream & ostr, const TStack<T>&stack) noexcept {

for (size\_t i = 0; i < stack.size; i++) {

ostr << stack.mas[i] << std::endl;

}

return ostr;

}

friend std::istream& operator>>(std::istream & istr, const TStack<T>&stack) noexcept {

for (size\_t i = 0; i < stack.size; i++) {

istr >> stack.mas[i];

}

return istr;

}

};

1. Приложение 2

stack\_sample.h

#include <iostream>

#include "MyStack.h"

using namespace std;

int main(int argc, char\*\* argv) {

setlocale(LC\_ALL, "Russian");

size\_t size, op;

int el;

try {

cout << "Введите размер стека: ";

cin >> size;

TStack<int> s(size);

while (true) {

cout << "Меню:\n"

<< "1. Добавить в стек\n"

<< "2. Удалить элемент\n"

<< "3. Верхний элемент\n"

<< "4. Количество элементов\n"

<< "5. Размер\n"

<< "Ввести операцию: ";

cin >> op;

cout << endl;

switch (op) {

case 1:

cout << "Введите элемент (int): ";

cin >> el;

s.Push(el);

break;

case 2:

s.Get();

break;

case 3:

cout << s.TopView() << endl;

break;

case 4:

cout << s.GetTop() << endl;

break;

case 5:

cout << s.GetSize() << endl;

break;

default:

break;

}

cout << s << endl;

}

}

catch (exception e) {

cerr << e.what() << endl;

}

}

1. Приложение 3

Test\_main.cpp  
#include <gtest.h>

int main(int argc, char \*\*argv) {

::testing::InitGoogleTest(&argc, argv);

return RUN\_ALL\_TESTS();

}//инициализация и запуск тестов, используя библиотеку Google Test

1. Приложение 4

Test\_stack.cpp  
#include "MyStack.h"

#include <gtest.h>

TEST(TStack, can\_create)

{

ASSERT\_NO\_THROW(TStack<int> s1(10));

}

TEST(TStack, can\_get\_size)

{

TStack<int> s1(2);

ASSERT\_EQ(2, s1.GetSize());

}

TEST(TStack, can\_get\_top)

{

TStack<int> s1(2);

ASSERT\_EQ(0, s1.GetTop());

}

TEST(TStack, can\_push\_element)

{

TStack<int> s1(1);

ASSERT\_NO\_THROW(s1.Push(10));

}

TEST(TStack, cant\_push\_element\_when\_stack\_is\_full)

{

TStack<int> s1(1);

s1.Push(10);

ASSERT\_ANY\_THROW(s1.Push(10));

}

TEST(TStack, can\_pop\_element)

{

TStack<int> s1(1);

s1.Push(10);

ASSERT\_NO\_THROW(s1.Get());

}

TEST(TStack, cant\_pop\_element\_when\_stack\_is\_empty)

{

TStack<int> s1(1);

ASSERT\_ANY\_THROW(s1.Get());

}

TEST(TStack, can\_get\_top\_element)

{

TStack<int> s1(1);

s1.Push(10);

ASSERT\_EQ(10, s1.TopView());

}

TEST(TStack, can\_check\_is\_empty)

{

TStack<int> s1(2);

ASSERT\_EQ(true, s1.IsEmpty());

s1.Push(1);

ASSERT\_EQ(false, s1.IsEmpty());

}

TEST(TStack, can\_check\_is\_full)

{

TStack<int> s1(1);

ASSERT\_EQ(false, s1.IsFull());

s1.Push(1);

ASSERT\_EQ(true, s1.IsFull());

}

1. Приложение 5

Postfix.h  
#pragma once

#include "../../stack/include/MyStack.h"

#include <string>

using namespace std;

class TArithmeticExpression

{

private:

string infix;

string postfix;

void ToPostfix();

int GetPriority(char op); // возвращает приоритет оператора

public:

TArithmeticExpression(string infix); // принимает арифметическое выражение в инфиксной форме и сохраняет его в поле infix

string GetInfix() { return infix; }

string GetPostfix() { return postfix; }

double Calculate();

};

1. Приложение 6

Postfix.cpp  
TArithmeticExpression::TArithmeticExpression(string \_infix)

{

infix = \_infix;

ToPostfix();

}

void TArithmeticExpression::ToPostfix() {

TStack<char> stack(200);

char stackItem;

string number = "";

for (char item : infix) {

if (item == '(') {

if (number != "") {

postfix += number + " ";

number = "";

}

stack.Push(item);

}

else if (item == ')') {

if (number != "") {

postfix += number + " ";

number = "";

}

stackItem = stack.Get();

while (stackItem != '(') {

postfix += stackItem;

postfix += " ";

stackItem = stack.Get();

}

}

// Встретился оператор

else if (item == '+' || item == '-' || item == '\*' || item == '/') {

if (number != "") {

postfix += number + " ";

number = "";

}

while (!stack.IsEmpty()) {

stackItem = stack.Get();

if (GetPriority(item) <= GetPriority(stackItem)) {

postfix += stackItem;

postfix += " ";

}

else {

stack.Push(stackItem);

break;

}

}

stack.Push(item);

}

// Встретилась часть числа (цифра или точка)

else {

number += item;

}

}

if (number != "") {

postfix += number + " ";

number = "";

}

while (!stack.IsEmpty()) {

stackItem = stack.Get();

postfix += stackItem;

postfix += " ";

}

}

double TArithmeticExpression::Calculate() {

double leftOperand, rightOperand;

TStack<double> stack(200);

string number = "";

for (char item : postfix)

{

if (item == ' ') {

if (number != "") {

stack.Push(stod(number)); // stod - string to double

number = "";

}

continue;

}

// Встретился оператор

if (item == '+' || item == '-' || item == '\*' || item == '/') {

double res = 0;

rightOperand = stack.Get();

leftOperand = stack.Get();

if (item == '+') res = leftOperand + rightOperand;

if (item == '-') res = leftOperand - rightOperand;

if (item == '\*') res = leftOperand \* rightOperand;

if (item == '/') {

if (rightOperand == 0)

throw - 1;

res = leftOperand / rightOperand;

}

stack.Push(res);

}

// Встретилась часть числа (цифра или точка)

else {

number += item;

}

}

return stack.TopView();

}

int TArithmeticExpression::GetPriority(char op) {

/\*

Получить приоритет операции

\*/

switch (op) {

case '(':

return 0;

case ')':

return 0;

case '+':

return 1;

case '-':

return 1;

case '/':

return 2;

case '\*':

return 2;

}

}

1. Приложение 7

MyForm.cpp  
#include "MyForm.h"

using namespace System;

using namespace System::Windows::Forms;

[STAThread]

void Main()

{

Application::EnableVisualStyles();

Application::SetCompatibleTextRenderingDefault(false);

CppWinForm1::MyForm form;

Application::Run(% form);

}

1. Приложение 8

Postfix\_sample.cpp  
#include <iostream>

#include "Postfix.h"

using namespace std;

int main()

{

setlocale(LC\_ALL, "Russian");

try {

string input;

cout << "Введите выражежение: ";

cin >> input;

TArithmeticExpression expression(input);// передается значение арифметического выражения

cout << expression.GetInfix() << endl;

cout << expression.GetPostfix() << endl;

cout << expression.Calculate() << endl;

}

catch (exception e) {

cout << "Ошибка" << endl;

}

}

1. Приложение 9

test\_expression.cpp  
#include "Postfix.h"

#include <gtest.h>

TEST(TArithmeticExpression, can\_get\_infix)

{

string expr = "a+b";

TArithmeticExpression expression(expr);

string str = "a+b";

ASSERT\_EQ(str, expression.GetInfix());

}

TEST(TArithmeticExpression, can\_get\_postfix)

{

string expr = "a+b";

TArithmeticExpression expression(expr);

string str = "a b + ";

ASSERT\_EQ(str, expression.GetPostfix());

}

TEST(TArithmeticExpression, can\_calculate)

{

string expr = "1+1";

TArithmeticExpression expression(expr);

ASSERT\_EQ(2, expression.Calculate());

}

TEST(TArithmeticExpression, example)

{

string expr = "(4+11-8/2\*(7\*3+4-7))\*3";

TArithmeticExpression expression(expr);

EXPECT\_EQ(-171.0, expression.Calculate());

}

TEST(TArithmeticExpression, priority)

{

string expr = "2+2\*2/2";

TArithmeticExpression expression(expr);

EXPECT\_EQ(4, expression.Calculate());

}