Министерство науки и высшего образования Российской Федерации

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

Институт Информационных технологий, математики и механики

Отчёт по учебной практике

**Вычисление арифметических выражений**

**Выполнил:**

студент гр. 381806-1

КасьянычевМихаил Петрович

**Проверил:**

кандидат технических наук, доцент кафедры МОСТ ИИТММ

Кустикова Валентина Дмитриевна

Нижний Новгород

2019 г.

Оглавление

[Оглавление 2](#_Toc34725613)

[Введение 3](#_Toc34725614)

[Постановка задачи 4](#_Toc34725615)

[Руководство пользователя 5](#_Toc34725616)

[Руководство программиста 6](#_Toc34725617)

[Структура программы 6](#_Toc34725618)

[Описание структур данных 6](#_Toc34725619)

[Класс Stack 6](#_Toc34725620)

[Класс Calculator 7](#_Toc34725621)

[Описание алгоритма 8](#_Toc34725622)

[Класс Stack 8](#_Toc34725623)

[Преобразование выражения в постфиксную форму 8](#_Toc34725624)

[Значения операндов и подсчет итогового значения 9](#_Toc34725625)

[Заключение 10](#_Toc34725626)

[Литература 11](#_Toc34725627)

[Приложения 12](#_Toc34725628)

[Приложение 1. Основная функция 12](#_Toc34725629)

[Приложение 2. Класс Stack 12](#_Toc34725630)

[Приложение 3. Класс Calculator 14](#_Toc34725631)

Введение

Целью лабораторной работы является изучение и практическое применение структуры Стек. Темой данной работы является разработка программы, позволяющей вычислять значения произвольного арифметического выражения при помощи структуры Стек.

Постановка задачи

Разработать программу для вычисления значения арифметического выражения при введенном выражении и значениях переменных.

Входные данные:

1. Строка с арифметическим выражением.
2. Значения переменных.

Выходные данные:

1. Постфиксная форма выражения.
2. Значение выражения.

Руководство пользователя

После запуска программы откроется интерфейс с предложением ввести арифметическое выражение (Рис. 1).



Рис. . Запрос на ввод выражения

После введения выражения нажмите Enter. Если введенное выражение корректно, программа выведет постфиксную форму выражения и предложит ввести значения указанных переменных (Рис. 2).

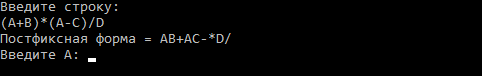


Рис. . Программа вывела постфиксную форму и ожидает ввода переменной А

После ввода значений всех переменных программа посчитает значение выражения и выведет результат (Рис. 3).

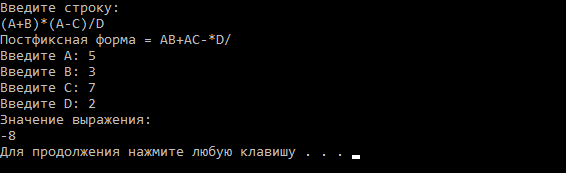


Рис. . Программа вывела значение выражения

Если вы введете некорректное выражение или значение переменной, при которой значение выражения не вычисляется, программа выдаст ошибку и завершит работу (Рис. 4), (Рис. 5).

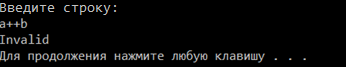


Рис. . Введено неверное выражение

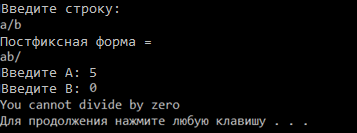


Рис. . Попытка деления на ноль

Руководство программиста

Структура программы

Программа состоит из 4-х файлов:

1. Main.cpp, который содержит функцию main.
2. Stack.h, который содержит класс Stack.
3. Calculator.h, который содержит класс Calculator.
4. Calculator.cpp, который содержит реализацию функций класса Calculator.

Описание структур данных

Класс Stack

Stack – класс, реализующий функционал структуры данных стек.

**Объявление:**

template<class ValueType>

classTStack {

public:

ValueType\* arr;

int size;

int top;

public:

TStack(int size = 10);

TStack(constTStack&);

~TStack();

void Push(ValueType);

void Pop();

boolIsEmpty() const;

boolIsFull() const;

ValueTypeTop() const;

};

**Описание методов класса:**

TStack(int size = 10)

Назначение: конструктор, инициализация объекта..

Входные параметры: размер стека.

Возвращаемое значение: нет.

TStack(const TStack&)

Назначение: конструктор копирования, создание копии текущего стека.

Входные параметры: константная ссылка на копируемый объект.

Возвращаемое значение: нет.

~TStack()

Назначение: деструктор, высвобождение памяти и удаление объекта класс.

Входные параметры: нет.

Возвращаемое значение: нет.

void Push(ValueType)

Назначение: добавление значения в стек.

Входные параметры: добавляемое значение.

Возвращаемое значение: нет.

void Pop()

Назначение: извлечение элемента из стека.

Входные параметры: нет.

Возвращаемое значение: извлекаемое значение.

bool IsEmpty() const

Назначение: проверка стека на пустоту.

Входные параметры: нет.

Возвращаемое значение: логическая переменная.

bool IsFull() const

Назначение: проверка стека на полноту.

Входные параметры: нет.

Возвращаемое значение: логическая переменная.

ValueType Top() const

Назначение: просмотр значения на вершине стека.

Входные параметры: нет.

Возвращаемое значение: значение на вершине стека.

Класс Calculator

Calculator – класс, реализующий методы для работы с арифметическими выражениями.

**Объявление**

static class Calculator {

private:

staticboolPrior(char, char);

static double Calc(double, double, char);

public:

static string CreatePostForm(const string&);

static double Calculate(const string&, double\*, int);

static double\* GetOper(const string&, int\*);

};

**Описание методов класса**

static string CreatePostFixForm(const string&)

Назначение: создание постфиксной формы.

Входные параметры: исходное выражение.

Возвращаемое значение: преобразованное в польскую нотацию исходное выражение.

static double Calculate(const string&, double\*, int)

Назначение: вычисление выражения при помощи польской нотации.

Входные параметры: исходное выражение, записанное в польской нотации, массив значений переменных, количество переменных.

Возвращаемое значение: результат вычисления выражения.

staticdouble\* GetValueOperands(conststring&, int\*)

Назначение:получение значений переменных.

Входные параметры: исходное выражение, записанное в польской нотации, ссылка на переменную которая содержит количество переменных.

Возвращаемое значение: массив значений переменных.

staticboolPriority(char, char)

Назначение: сравнение приоритета пришедшей операции и операции на вершине стека.

Входные параметры: 2 значения операций.

Возвращаемое значение: логическая переменная.

staticdoubleCalculator(double, double, char)

Назначение: калькулятор.

Входные параметры: значения переменных и арифметическое действие в типе char.

Возвращаемое значение: результат вычисления.

Описание алгоритма

Класс Stack

Стек (англ. stack — стопка; читается стэк) — абстрактный тип данных, представляющий собой список элементов, организованных по принципу LIFO (англ. lastin — firstout, «последнимпришёл — первымвышел»).

Характеристики стека:

* Массив данных.
* Индекспоследнего занятого.
* Размер стека.

Методы стека:

**Добавление элемента в стек:** если стек не полный, то увеличить значение индекса последнего не занятого на единицу и положить по этому индексу элемент в массив данных.

**Получение данных на вершине стека:** если стек не пустой, то возвращается значение по индексу последнего занятого.

**Удаление данных:** если стек не пустой, то возвращается значение по индексу последнего занятого и индекс последнего занятого уменьшается на единицу.

**Проверка на пустоту:** если значение последнего занятого равно -1, то стек пуст.

**Проверка на полноту:** если значение последнего занятого равно размеру стека – 1, то стек полон.

Преобразование выражения в постфиксную форму

1. Создаются 2 стека: постфиксная форма и стек операторов.
2. Поочередно просматривается каждый символ арифметического выражения.
3. Если символ является арифметическим знаком («+», «-», «\*», «/») или знак открывающейся скобки «(», он отправляется в стек операторов.
4. Если символ является закрывающейся скобкой «)», достаются все элементы из стека операторов до «(» и добавляются в стек постфиксной формы.
5. Если символ операнд, он добавляется в стек постфиксной формы.
6. После завершения цикла все элементы стека операторов перекладываются в стек постфиксной формы.
7. Все элементы стека постфиксной формы добавляются в итоговую строку, разделяясь пробелами(

Таблица 1. Пример преобразования в постфиксную форму

Исходная строка: A+(B-С)\*A.

|  |  |  |
| --- | --- | --- |
| Элемент строки | Постфиксная форма | Стек операторов |
| A | A |  |
| + | A | + |
| ( | A | +( |
| B | AB | +( |
| - | AB | +(- |
| С | ABС | +(- |
| ) | ABС- | + |
| \* | ABС- | +\* |
| A | ABС-A | +\* |
|  | ABС-A\*+ |  |

Итоговая строка: ABС–A\*+.

Значения операндов и подсчет итогового значения

1. Поочередно просматриваются все элементы постфиксной формы, считается количество операндов.
2. Выделяется память для 2 массивов: массив названий переменных и массив их значений.
3. Далее будет выведен запрос на значение операнда, считывание и запись в массив.
4. Создается стек итогового значения.
5. Поочередно просматриваются все элементы постфиксной формы.
6. Если элемент является операндом, его значение отправляется в итоговый стек.
7. Если элемент является оператором, из итогового массива достаются 2 последних элемента, производится необходимая арифметическая операция и значение отправляется обратно в итоговый стек.
8. Последнее значение в итоговом стеке является итоговым для всего выражения.

Таблица2. Пример запоминания значения операндов

|  |  |  |  |
| --- | --- | --- | --- |
| Массив переменных | А | В | 5 |
| Массив значений | 10 | 15 | 5 |

Таблица3. Пример подсчета итогового значения

|  |  |
| --- | --- |
| Элемент постфиксной формы | Итоговый стек |
| A | 10 |
| B | 10 15 |
| 5 | 10 15 5 |
| - | 10 10 |
| A | 10 10 10 |
| \* | 100 |
| + | 110 |

Итоговое значение выражения 110.

Заключение

Разработана программа для вычисления значения арифметического выражения при введенном выражении и значениях переменных с выводом постфиксной формы выражения.

Литература

Горьков Алексей Обратная польская запись:[<https://habr.com/ru/post/100869/>] 03.08.2010.

Приложения

Приложение 1. Основная функция

void main()

{

setlocale(LC\_ALL, "Russian");

string s, rez;

int h = 1;

double PFV = 0;

cout << "Введите строку:" << endl;

cin >> s;

try

{

rez = Calculator::CreatePostForm(s);

}

catch(const char \*osh)

{

cout << osh << endl;

system("pause");

return;

}

cout << "Постфиксная форма = ";

cout << rez << endl;

double\* ex;

char\* oper;

try

{

Calculator::GetOper(rez, ex, oper, h);

}

catch (const char\* osh)

{

cout << osh << endl;

system("pause");

return;

}

try

{

PFV = Calculator::Calculate(rez, ex, oper, h);

}

catch (const char\* osh)

{

cout << osh << endl;

system("pause");

return;

}

cout << "Значение выражения: " << endl;

cout << PFV << endl;

system("pause");

}

Приложение 2. Класс Stack

template<classValueType>

classTStack

{

private:

int size;

ValueType \*arr;

int top;

public:

TStack(int\_size = 10);

TStack(constTStack<ValueType>&tmp);

~TStack();

voidPush(ValueTypec);

ValueTypePop();

ValueTypeTop() const;

TStack&operator=(constTStack&tmp);

boolIsFull()const;

boolIsEmpty()const;

};

template<classValueType>

TStack<ValueType>::TStack(int\_size)

{

size = \_size;

top = -1;

arr = newValueType[size];

}

template<classValueType>

TStack<ValueType>::TStack(constTStack<ValueType>&tmp)

{

size = tmp.size;

top = tmp.top;

arr = newValueType[size];

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

arr[i] = tmp.arr[i];

}

template<classValueType>

TStack<ValueType>::~TStack()

{

size = 0;

top = -1;

delete[] arr;

}

template<classValueType>

void TStack<ValueType>::Push(ValueTypec)

{

if (this->IsFull())

throw"Full";

arr[++top] = c;

}

template<classValueType>

void TStack<ValueType>::Pop()

{

if (this->IsEmpty())

throw"Empty";

top--;

}

template<classValueType>

ValueType TStack<ValueType>::Top() const

{

return arr[top];

}

template<classValueType>

TStack<ValueType>&TStack<ValueType>::operator=(constTStack<ValueType>&tmp)

{

size = tmp.size;

top = tmp.top;

delete[] arr;

arr = newValueType[size];

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

{

arr[i] = tmp.arr[i];

}

return \*this;

}

template<classValueType>

boolTStack<ValueType>::IsFull() const

{

return (top == size);

}

template<classValueType>

boolTStack<ValueType>::IsEmpty() const

{

return (top == -1);

}

Приложение 3. Класс Calculator

Bool Calculator::Prior(chara, charb)

{

if ((a == '\*' || a == '/') && (b == '\*' || b == '/')) returntrue;

if ((a == '+' || a == '-') && (b == '+' || b == '-')) returntrue;

if ((a == '+' || a == '-') && (b == '\*' || b == '/')) returntrue;

returnfalse;

}

double Calculator::Calc(doublea, doubleb, charc)

{

switch (c)

{

case'+':

returna + b;

case'-':

returna - b;

case'\*':

returna \* b;

case'/':

if (b == 0) throw"You cannot divide by zero";

returna / b;

}

}

string Calculator::CreatePostForm(const string& s)

{

int left = 0;

int right = 0;

for (int i = 0; i < s.length(); i++) {

if (s[i] == '(') left++;

if (s[i] == ')') right++;

}

if (left != right) throw "Invalid";

if ((s[0] == '+') || (s[0] == '-') || (s[0] == '\*') || (s[0] == '/')) throw "Invalid";

if (s[s.length() - 1] == '+' || s[s.length() - 1] == '-' || s[s.length() - 1] == '\*' || s[s.length() - 1] == '/') throw "Invalid";

for (int i = 0; i < s.length(); i++)

if ((s[i] == '1') || (s[i] == '2') || (s[i] == '3') || (s[i] == '4') || (s[i] == '5') || (s[i] == '6') || (s[i] == '7') || (s[i] == '8') || (s[i] == '9') || (s[i] == '0'))

throw "Invalid";

for (int j = 1; j < s.length(); j++)

if ((s[j] != '+') && (s[j] != '-') && (s[j] != '\*') && (s[j] != '/') && (s[j] != '(') && (s[j] != ')') && (s[j - 1] != '+') && (s[j - 1] != '-') && (s[j - 1] != '\*') && (s[j - 1] != '/') && (s[j - 1] != '(') && (s[j - 1] != ')'))

throw "Invalid";

for (int i = 1; i < s.length(); i++)

if (((s[i] == '+') || (s[i] == '-') || (s[i] == '\*') || (s[i] == '/')) && ((s[i - 1] == '+') || (s[i - 1] == '-') || (s[i - 1] == '\*') || (s[i - 1] == '/')))

throw "Invalid";

TStack<double> operands(s.length());

TStack<double> operators(s.length());

string PFForm = "";

for (int i = 0; i < s.length(); i++)

{

bool f = (s[i] == '+') || (s[i] == '-') || (s[i] == '\*') || (s[i] == '/') || (s[i] == '(') || (s[i] == ')');

if (!f)

{

operands.Push(s[i]);

continue;

}

if (f)

{

if (s[i] == ')')

{

while (operators.Top() != '(')

{

operands.Push(operators.Top());

operators.Pop();

}

operators.Pop();

continue;

}

if (Prior(s[i], operators.Top()))

{

while ((Prior(s[i], operators.Top())) && !(operators.IsEmpty()) && (operators.Top() != '('))

{

operands.Push(operators.Top());

operators.Pop();

}

operators.Push(s[i]);

continue;

}

operators.Push(s[i]);

}

}

while (!operators.IsEmpty())

{

operands.Push(operators.Top());

operators.Pop();

}

while (!operands.IsEmpty())

{

operators.Push(operands.Top());

operands.Pop();

}

while (!operators.IsEmpty())

{

PFForm += operators.Top();

operators.Pop();

}

return PFForm;

}

double Calculator::Calculate(const string& s, const double\* a, const char\* c, int h)

{

TStack<double> L(h);

for (int i = 0; i < s.length(); i++)

{

if ((s[i] == '+') || (s[i] == '-') || (s[i] == '\*') || (s[i] == '/'))

{

double first = L.Top();

L.Pop();

double second = L.Top();

L.Pop();

L.Push(Calc(second, first, s[i]));

}

else

{

for (int j = 0; j < h; j++)

if (c[j] == s[i])

L.Push(a[j]);

}

}

return L.Top();

}

void Calculator::GetOper(const string& s, double\*& a, char\*& c, int& h)

{

h = 0;

for (int i = 0; i < s.length(); i++)

if ((s[i] != '\*') && (s[i] != '/') && (s[i] != '+') && (s[i] != '-'))

h++;

a = new double[h];

c = new char[h];

int P = 0;

int W;

for (int i = 0; i < s.length(); i++)

{

W = 0;

if (s[i] != '\*' && s[i] != '/' && s[i] != '+' && s[i] != '-')

{

for (int j = 0; j < P; j++) if (c[j] == s[i])

{

W = 1;

break;

}

if (W == 0)

{

c[P] = s[i];

cout << "Введите " << s[i] << ": ";

cin >> a[P];

P++;

}

}

}

}