Федеральное государственное автономное образовательное учреждение

высшего образования

«Нижегородский государственный университет

им. Н.И. Лобачевского»

Институт Информационных Технологий, Математики и Механики

Отчёт по лабораторной работе

Вычисление арифметических выражений

Выполнил:

студент ф-та ИИТММ гр. 0823-1

Ванеев Н.Ю.

Проверил:

к.т.н., ассистент каф. ПрИнж ИИТММ

Сиднев А.А.

Нижний Новгород

2016 г.

Содержание

[Введение 3](file:///D:\plak\3laba\Отчет.docx#_Toc405754030)

[Постановка задачи 4](file:///D:\plak\3laba\Отчет.docx#_Toc405754031)

[Руководство пользователя 5](file:///D:\plak\3laba\Отчет.docx#_Toc405754032)

[Руководство программиста 6](file:///D:\plak\3laba\Отчет.docx#_Toc405754033)

[Описание структуры программы 6](file:///D:\plak\3laba\Отчет.docx#_Toc405754034)

[Описание структур данных 6](file:///D:\plak\3laba\Отчет.docx#_Toc405754035)

[Описание алгоритмов 8](file:///D:\plak\3laba\Отчет.docx#_Toc405754036)

[Заключение 9](file:///D:\plak\3laba\Отчет.docx#_Toc405754037)

[Литература 10](file:///D:\plak\3laba\Отчет.docx#_Toc405754038)

[Приложения 11](file:///D:\plak\3laba\Отчет.docx#_Toc405754039)

[Приложение 1 11](file:///D:\plak\3laba\Отчет.docx#_Toc405754040)

[Приложение 2 11](file:///D:\plak\3laba\Отчет.docx#_Toc405754041)

[Приложение 3 12](file:///D:\plak\3laba\Отчет.docx#_Toc405754042)

# Введение

Выполненная лабораторная работа направлена на изучение стека и работу с ним, тренировку навыков описания классов, реализации алгоритмов, таких как проверка корректности алгебраического выражения, перевод выражения в постфиксную ("польскую") запись, вычисление значения алгебраического выражения.

В нашей жизни невозможно обходиться без вычислений. Они применяются практически во всех сферах жизни. Калькуляторы и программы, вычисляющие значения арифметических выражений, используются очень часто. Разобраться, как происходят вычисления изнутри, важно для программиста. Использование стека для хранения операндов и операторов, позволяет не только разобраться с принципом вычисления арифметических выражений, но и узнать возможности применения стека, убедиться в его полезности.

Написание работ такого типа важно для программиста, так как часто возникает необходимость хранить данные в виде стека. Реализованный нами алгоритм разбора арифметического выражения является одним из примеров использованию стека.

# Постановка задачи

Необходимо реализовать программу, выполняющую вычисление арифметического выражения с вещественными числами. Выражение может содержать:

* переменные;
* вещественные числа;
* операции +, -, /, \*,^,||;
* унарный "-" в начале выражения или после открывающей скобки.

Тесты проверяют введенное выражение на корректность:

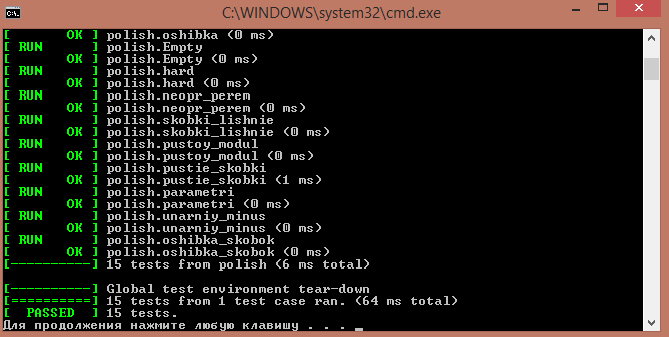
* соответствие скобок;
* наличие недопустимых знаков;
* недопустимый порядок операций (например, отсутствие операнда между знаками операций).

Так же тесты вводят значения переменных, и получает вычисленное значение выражения.

# Руководство пользователя

Пример работы программы.

Программа содержит 15 тестов для проверки правильности работы



Для подсчёта выражений используется класс Polish

Public методы этого класса:

public:

char\* Pols;

Polish(char\*);

void AddX(char, int);

double Culc();

};

Метод Polish(char\*) – это конструктор польской записи. На вход он принимает строку и преобразует ее в обратную польскую запись для дальнейших вычислений.

void AddX(char, int) – метод, позволяющий задать переменные

double Culc() – метод, который вычисляет значение выражения

Руководство программиста

## Описание структуры программы

Программа состоит из следующих составных частей:

* Заголовочный файл Staсk.h.

Cодержит описание класса Staсk*,* реализующего структуру данных стек, конструктор, деструктор, функции, необходимые для работы со стеком, такие как контроль пустоты стека, извлечение элемента из стека, добавление элемента в стек, чтение верхнего элемента.

* Заголовочный файл Polish.h.

Содержит описание класса Polish, реализующего операции с алгебраическим выражением, прототипы функций.

* Модуль Polish.cpp , который содержит реализацию необходимых функций для работы с алгебраическим выражением;
* Модуль test.cpp.

Данный модуль содержит тесты для проверки программы.

## Описание структур данных

В программе реализована структура данных — стек.

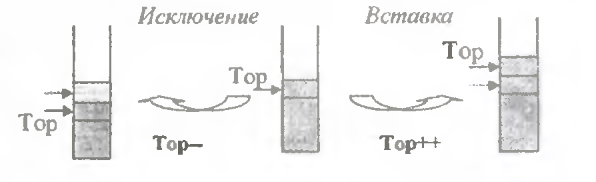
Стек — структура данных, в которой добавление элементов в который и выборка из которого выполняются с одного конца, называемого вершиной стека. Другие операции со стеком не определены. При выборке элемент исключается из стека. Стек реализует принцип обслуживания LIFO (last in — first out, последним пришел — первым ушел). Стеки широко применяются в системном программном обеспечении, компиляторах, в различных рекурсивных алгоритмах.

Стек характерен тем, что получить доступ к его элементам можно лишь с одного конца, называемого вершиной стека. Стек часто сравнивают со стопкой книг, где чтобы воспользоваться одной из них, и не нарушить установленный порядок, нужно поднять все те книги, что лежат выше нее, а положить книгу можно лишь поверх всех остальных.

В программе стек представляет собой массив, элементы которого — шаблонный класс. Основные операции над стеками:

* вставка элемента;
* исключение элемента;
* чтение верхнего элемента.

Схематично изобразить первые две операции можно следующим образом, где *Top* —верхний элемент списка:



При вставке элемента добавленный элемент становится вершиной, а при удалении вершиной будет являться элемент, следующий за старой вершиной.

В классе *Staсk* имеются следующие поля и методы:

TNode<T>\* top; // указатель на вершину

~Stak(); // деструктор

bool Empty(); // контроль пустоты

bool Pop(); // извлечение из стека

TNode<T> Top(); // чтение верхнего элемента в стеке

void Push(T); // добавление в стек

T& gettop(); // взятие верхнего элемента

Поля и методы класса *Polish*

## Perem\* x ; // хранение значения переменной

## int prioritet(char); // приоритет операций

## bool isoperation(char); // определение операций

## bool isStr(char); // определение символов

## CNode\* perevod(char\*); // разбор выражения

## public:

## char\* Pols;

## Polish(char\*);

## void AddX(char, int);

## double Culc();

## };

## Описание алгоритмов

В написанной программе осуществлены следующие алгоритмы:

Алгоритм Разбиения выражений на лексемы (метод perevod), (Приложение 1)

1.Мы просматриваем i-ый символ строки.

1.2 Если символ – цифра или буква, то добавляем его значение к созданной строке, ,если символ допустим для алфавита, добавляем его в список лексем

1.3 Если символ –операция, то создаём строку, записываем в неё значение иначе добавляем его в список лексем.

Алгоритм перевода в польскую запись (метод Polish) (Приложение 2)

1.проходим по лексемам и смотрим их значения

2.1. если встречается переменная, то записываем ее сразу польскую запись

2.2 если встречается число так же записываем его в польскую запись

2.3. если встречается операция и ее приоритет больше чем приоритет, лежащий в стеке операции то она просто кладется в стек

2.4 если встречается операция и ее приоритет меньше чем приоритет, лежащий в стеке операции то операции изымаются из стека и записываются в польскую запись пока приоритет операций не станет верным.

1. Алгоритм вычисления значения выражения (метод Culc) (Приложение 3)

3.1. Выполняем перевод записи алгебраического выражения в постфиксную форму

3.2. Проходя по постфиксной записи, встречая число, кладем его в стек

3.3. Встречая переменную, также кладем его в стек

3.4. Встречая оператор "+"," -"," /" ,"\*",”^”,|| достаем нужное количество для данной операции чисел или переменных из стека и выполняем соответствующую операцию. Результат выполненной операции кладем обратно в стек

3.5. В итоге в стеке останется вычисленное значение выражения. Оно возвращается.

# Заключение

В выполненной лабораторной работе была реализована структура данных — стек, с помощью него получена постфиксная запись алгебраического выражения, выполнено вычисление арифметического выражения с вещественными числами. Выражение может содержать:

* переменные;
* вещественные числа;
* операции +, -, /, \*,^,||;
* унарный "-" в начале выражения или после открывающей скобки.

А также контролировалась корректность введенного выражения:

* соответствие скобок;
* наличие недопустимых знаков;
* недопустимый порядок операций (например, отсутствие операнда между знаками операций).

Таким образом, в ходе выполнения лабораторной работы был изучена структура данных — стек, получены навыки работы с ним, закреплены умения работы с классами, реализованы алгоритмы, такие как проверка корректности алгебраического выражения, перевод выражения в постфиксную ("польскую") запись, вычисление значения алгебраического выражения.

# Литература

1. Гергель В.П. Методы программирования. – Нижний Новгород, 2002. – Презентация.
2. Павловская Т.А. C/C++. Программирование на языке высокого уровня. – СПб.: Питер, 2003. – 120с.:ил.

# Приложения

## Приложение 1

Реализация алгоритма разбиения на лексемы

CNode\* Polish::perevod(char\* s)

{

int pos;

CNode\* Perevod = new CNode;

CNode\* help = Perevod;

char \* str = new char[10];

int len = 0, hp = 0;

char\* a;

if (!s[0])

throw std::logic\_error("stroka empty");

int sl = strlen(s);

char \*exp = new char[sl + 2];

exp = strcpy(exp, s);

exp[sl] = ' ';

exp[sl + 1] = 0;

if (isStr(exp[0]))

pos = 0;

else

pos = 1;

for (int i = 0; exp[i] != 0; i++)

{

a = new char(exp[i]);

if (pos == 0)

if (isStr(\*a) || ((\*a == '.') && (!hp)))

{

if (\*a == '.')

hp = 1;

str[len] = \*a;

len++;

delete a;

}

else

{

if (!isoperation(\*a))

throw std::logic\_error("oshibka!");

str[len] = 0;

help = Add(help, str);

pos = 1;

str = new char[10];

len = 0;

hp = 0;

if (\*a != ' ')

help = Add(help, a);

continue;

}

if (pos == 1)

if (isStr(\*a)) {

str[len] = \*a;

len++;

pos = 0;

delete a;

}

else {

if (!isoperation(\*a))

throw std::logic\_error("oshibka");

if (\*a != ' ') {

help = Add(help, a);

}

}

}

delete[]exp;

Perevod = Perevod->next;

return Perevod;

}

## Приложение 2

Реализация алгоритма перевода в польскую форму записи:

Polish::Polish(char\* exp)

{

x = 0;

char\* res = new char[strlen(exp) \* 2];

int pos = 0;

int mod = 0, br = 0, clmod = 0;

char last = ' ';

CNode\* parse = perevod(exp);

Stack<char> st;

if (parse == 0)

throw std::logic\_error("oshibka");

while (parse)

{

char \* el = parse->data;

if (isStr(\*el))

{

for (int i = 0; el[i] != 0; i++, pos++)

res[pos] = el[i];

res[pos] = ' ';

pos++;

}

else

{

switch (\*el)

{

case '(':

st.push(\*el);

br = 1;

break;

case '|':

if (mod || !pos || (isoperation(last) && (last != '|')))

{

st.push(\*el);

if (parse->next)

if (\*parse->next->data == '|')

mod = 1;

else

mod = 0;

clmod = 1;

}

else

{

while (st.gettop() != '|')

{

res[pos++] = st.gettop();

st.pop();

}

res[pos++] = '|';

st.pop();

clmod = 0;

}

break;

case ')':

if (st.gettop() == '(')

throw std::logic\_error("oshibka");

if (br)

{

while (st.gettop() != '(') {

res[pos] = st.gettop();

pos++;

st.pop();

}

}

else

throw std::logic\_error("oshibka");

st.pop();

break;

default:

while (!st.empty())

{

if ((prioritet(\*el)) <= prioritet(st.gettop()))

{

res[pos] = st.gettop();

pos++;

st.pop();

}

else

break;

}

if (\*el == '-')

{

if (!pos)

st.push('\_');

else

{

if ((last == '|') && (clmod))

st.push('\_');

else

if (!st.empty())

{

if (last == '(')

{

st.push('\_');

}

else

st.push(\*el);

}

else

st.push(\*el);

}

}

else

st.push(\*el);

}

}

parse = parse->next;

last = \*el;

}

while (!st.empty()) {

if ((st.gettop() != '(') && (st.gettop() != '|'))

res[pos] = st.gettop();

else

throw std::logic\_error("oshibka");

pos++;

st.pop();

}

res[pos] = 0;

Pols = res;

}

***Приложение 3***

Реализация алгоритма вычисления значения выражения:

double Polish::Culc()

{

if (!Pols)

throw std::logic\_error("oshibka");

int len = strlen(Pols), slen;

double a, b;

Perem\* help;

Stack<double> st;

char\* num;

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

{

if ((Pols[i] >= 'a') && (Pols[i] <= 'z'))

{

help = x;

while (help)

{

if (help->ch == Pols[i])

break;

help = help->next;

}

if (!help)

throw std::logic\_error("oshibka");

st.push(help->val);

i++;

}

else

if ((Pols[i] >= '0') && (Pols[i] <= '9'))

{

num = new char[10];

slen = 0;

while (Pols[i] != ' ')

{

if ((Pols[i] >= 'a') && (Pols[i] <= 'z'))

throw std::logic\_error("oshibka");

num[slen] = Pols[i];

slen++;

i++;

}

num[slen] = 0;

st.push(atof(num));

}

else

if (!st.empty())

{

b = st.gettop();

st.pop();

if ((Pols[i] != '|') && (Pols[i] != '\_'))

{

if (!st.empty())

{

a = st.gettop();

st.pop();

}

else

throw std::logic\_error("Wrong expression!");

}

switch (Pols[i])

{

case '+':

st.push(a + b);

break;

case '-':

st.push(a - b);

break;

case '/':

st.push(a / b);

break;

case '\*':

st.push(a \* b);

break;

case '^':

st.push(pow(a, b));

break;

case '|':

st.push(abs(b));

break;

case '\_':

st.push(-b);

break;

default:

throw std::logic\_error("Wrong expression!");

}

}

else

throw std::logic\_error("Wrong expression!");

}

double result = st.gettop();

st.pop();

if (st.empty())

return result;

else

throw std::logic\_error("Wrong expression!");

}