Министерство образования и науки 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. 381606-3

Волков А.А.

Проверил:

к.т.н., ст. преп. каф. МОСТ ИИТММ

Кустикова В.Д.

Нижний Новгород

2017 г.

Содержание

[Введение 3](#_Toc438456330)

[Постановка задачи 4](#_Toc438456331)

[Руководство пользователя 5](#_Toc438456332)

[Руководство программиста 6](#_Toc438456333)

[Описание структуры программы 6](#_Toc438456334)

[Описание структур данных 6](#_Toc438456335)

[Описание алгоритмов 6](#_Toc438456336)

[Заключение 7](#_Toc438456337)

# Введение

Лабораторная работа направлена на практическое освоение динамической структуры данных Стек. С этой целью в лабораторной работе изучаются различные варианты структуры хранения стеков и разрабатываются методы и программы решения ряда задач с использованием стеков. В качестве области приложений выбрана тема вычисления арифметических выражений, возникающей при трансляции программ на языке программирования высокого уровня в исполняемые программы. При вычислении произвольных арифметических выражений возникают две основные задачи: проверка корректности введённого выражения и выполнение операций в порядке, определяемом их приоритетами и расстановкой скобок. Существует алгоритм, позволяющий реализовать вычисление произвольного арифметического выражения за один просмотр без хранения промежуточных результатов. Для реализации данного алгоритма выражение должно быть представлено в постфиксной форме. Рассматриваемые в данной лабораторной работе алгоритмы являются начальным введением в область машинных вычислений.

# Постановка задачи

В рамках лабораторной работы ставится задача реализации программ, обеспечивающих поддержку стеков, и разработки программных средств, производящих обработку арифметических выражений, включая проверку правильности записи выражения, перевод в постфиксную форму и вычисление результата. В начальной – самой простой постановке – можно предполагать, что проверка записи выражения состоит в контроле правильности расстановки скобок, перевод в постфиксную форму производится только для корректных выражений, а вычисление – для корректных выражений, содержащих только числовые операнды и допустимые знаки операций.

# Руководство пользователя

1. Запустите программу.
2. В появившемся окне введите арифметическое выражение в инфиксной форме.(программа должна вывести выражение в постфиксной форме)
3. Введите в консоль значения операндов.(команда должна вывести значение выражения)

# Руководство программиста

# Описание структуры программы

1. ../include
   1. List.h  
      Содержит описание и реализацию шаблонного класса List
   2. Node.h  
      Содержит описание и реализацию класса Node
   3. Stack.h  
      Содержит описание и реализацию класса Stack
2. ../samples
   1. Main.cpp  
      Содержит основной код программы
3. ../src  
   Содержит файлы исходного кода
4. ../build  
   Содержит директорию с решением и проектом для MS Visual Studio

***Описание структур данных***

*Шаблонный класс List*

template <typename T>

class List

{

private:

Node<T> \*root;

public:

List();

List(const List<T>&);

~List();

void Add(T);

void Remove(T);

void InsertBefore(T, T);

void InsertAfter(T, T);

void InsertEnd(T);

void print() const;

int Size() const;

T GetRoot()const;

Node<T> \*Search(T);

};

*Описание методов:*

1. Node<TKey> \*root - указатель на начало списка;
2. Add - вставка ключа в начало;
3. Remove - удаление заданного ключа;
4. InsertBefore - вставка до элемента с заданным ключам;
5. InsertAfter - вставка после элемента с заданным ключам;
6. InsertEnd - вставка ключа в конец;
7. Print - печать списка;
8. int Size() const - определение длинны списка;

*Шаблонный класс Node*

template <typename TKey>

class Node

{

public:

TKey key;

Node<TKey> \*pNext;

};

*Шаблонный класс Stack*

template <typename TKey>

class Stack

{

private:

List<TKey>\* list;

public:

Stack();

Stack(const Stack<TKey>&);

~Stack();

void Push(TKey);

TKey Pop();

TKey Take() const;

bool IsEmpty() const;

bool IsFull() const;

};

template <typename TKey>

Stack<TKey>::Stack()

{

list = new List<TKey>;

};

template <typename TKey>

Stack<TKey>::Stack(const Stack&S)

{

list = new List<TKey>(S.list);

};

template <typename TKey>

Stack<TKey>::~Stack()

{

delete list;

};

template <typename TKey>

void Stack<TKey>::Push(TKey key)

{

if (IsFull()) throw "Is Full";

list->Add(key);

};

template <typename TKey>

TKey Stack<TKey>::Take()const

{

return list->GetRoot();

};

template <typename TKey>

TKey Stack<TKey>::Pop()

{

if (IsEmpty()) throw "Is Empty";

TKey key = list->GetRoot();

try {

list->Remove(key);

}

catch (const char\* ex)

{

cout << ex << endl;

}

return key;

};

template <typename TKey>

bool Stack<TKey>::IsEmpty() const

{

return(list->Size() == 0);

};

template <typename TKey>

bool Stack<TKey>::IsFull() const

{

TKey key = -1;

try {

list -> Add(key);

list->Remove(key);

}

catch (const char\* ex)

{

return true;

}

return false;

};

*Описание методов:*

1. Push - добавление элемента
2. Pop - изъятие элемента
3. Take – возвращает элемент, без его изъятия
4. IsEmpty – проверка на пустоту
5. IsFull – проверка на полноту

***Описание алгоритмов***

Данный алгоритм основан на использовании стека. На вход алгоритма поступает строка символов, на выходе должна быть получена строка с постфиксной формой. Каждой операции и скобкам приписывается приоритет.

1. «(» - 0;
2. «)» - 1;
3. «+ -» - 2;
4. « \* /» - 3;

Предполагается, что входная строка содержит синтаксически правильное выражение. Входная строка просматривается посимвольно слева направо до достижения конца строки. Операндами будем считать любую последовательность символов входной строки, не совпадающую со знаками определённых в таблице операций. Операнды по мере их появления переписываются в выходную строку. При появлении во входной строке операции, происходит вычисление приоритета данной операции. Знак данной операции помещается в стек, если:

1. Приоритет операции равен 0 (это « ( » );
2. Приоритет операции строго больше приоритета операции, лежащей на вершине стека;
3. Стек пуст.

В противном случае из стека извлекаются все знаки операций с приоритетом больше или равным приоритету текущей операции. Они переписываются в выходную строку, после чего знак текущей операции помещается в стек. Имеется особенность в обработке закрывающей скобки. Появление закрывающей скобки во входной строке приводит к выталкиванию и записи в выходную строку всех знаков операций до появления открывающей скобки. Открывающая скобка из стека выталкивается, но в выходную строку не записывается. Таким образом, ни открывающая, ни закрывающая скобки в выходную строку не попадают. После просмотра всей входной строки происходит последовательное извлечение всех элементов стека с одновременной записью знаков операций, извлекаемых из стека, в выходную строку. Описание алгоритмов, применяющихся в программе.

int functions::Priority(char sym)

{

if ((sym == '\*') || (sym == '/')) {

return 3;

}

if ((sym == '+') || (sym == '-')) {

return 2;

}

if (sym == ')') {

return 1;

}

if (sym == '(') {

return 0;

}

return -1;

}

string functions::postfix(string line)

{

Stack<char> S1;

Stack<char> S2;

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

{

if (((line[i] >= 'a') && (line[i] <= 'z')) || ((line[i] >= 'A') && (line[i] <= 'Z')))

S1.Push(line[i]);

else if (line[i] == ')')

{

while ((!S2.IsEmpty()) && (S2.Take() != '('))

{

S1.Push(S2.Pop());

}

if (!S2.IsEmpty() && (S2.Take() == '('))

S2.Pop();

else

throw "S2 is empty";

}

else if (line[i] == '(')

S2.Push(line[i]);

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

{

if (Priority(S2.Take()) <= Priority(line[i]) || S2.Take() == '(')

S2.Push(line[i]);

else

{

while ((!S2.IsEmpty()) && (Priority(S2.Take()) >= Priority(line[i])))

{

S1.Push(S2.Pop());

}

S2.Push(line[i]);

}

}

}

while (!S2.IsEmpty())

{

if((S2.Take()!=')')&&(S2.Take()!='('))

S1.Push(S2.Pop());

else

S2.Pop();

}

string postfix = "";

while (!S1.IsEmpty())

{

postfix = S1.Pop() + postfix;

}

return postfix;

}

string functions::operands(string line)

{

int n=0;

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

{

if(((line[i]>='A')&&(line[i]<='Z'))|| ((line[i] >= 'a') && (line[i] <= 'z')))

n++;

}

char\* arr=new char[n];

int j=0;

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

{

if(((line[i]>='A')&&(line[i]<='Z'))|| ((line[i] >= 'a') && (line[i] <= 'z')))

{

arr[j]=line[i];

j++;

}

}

int k=1;

for( int i=1;i<n;i++)

{

bool flag=true;

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

if (arr[i]==arr[j])

flag=false;

if (flag) k++;

}

char\* arr1=new char[k+1];

arr1[0]=arr[0];

int g=1;

for( int i=1;i<n;i++)

{

bool flag=true;

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

if (arr1[j]==arr[i])

flag=false;

if (flag) arr1[g++]=arr[i];

}

arr1[g] = '\0';

string op(arr1);

delete [] arr;

delete [] arr1;

return op;

}

NameValue \* functions::NVmas( string op)

{

NameValue \*NV = new NameValue[op.length()];

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

{

cout << "Enter the " << i+1 << " value: ";

cin >> NV[i].Value;

NV[i].Operand = op[i];

cout << endl;

}

return NV;

}

float functions::Result(string line, int sizeOfTotalExpression, NameValue\* NV)

{

Stack<float> S1;

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

{

int j = 0;

if (((line[i] >= 'A') && (line[i] <= 'Z')) || ((line[i] >= 'a') && (line[i] <= 'z')))

{

while (NV[j].Operand != line[i])

j++;

S1.Push(NV[j].Value);

}

float a, b;

if (line[i] == '+')

{

a = S1.Pop();

b = S1.Pop();

S1.Push(b + a);

}

if (line[i] == '-')

{

a = S1.Pop();

b = S1.Pop();

S1.Push(b - a);

}

if (line[i] == '\*')

{

a = S1.Pop();

b = S1.Pop();

S1.Push(b \* a);

}

if (line[i] == '/')

{

a = S1.Pop();

b = S1.Pop();

if(a!=0)

S1.Push(b / a);

else

throw " delenie na 0";

}

}

return S1.Pop();

}

# Заключение

В рамках лабораторной работы мы реализовали программу, обеспечивающую поддержку стеков, и разработали программные средства, производящие обработку арифметических выражений (перевод в постфиксную форму и вычисление результата).