

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

Каганов Д.А.

Проверил:
к.т.н., ст. преп. каф. МОСТ ИИТММ

Кустикова В.Д.

Нижний Новгород
2017 г.

Содержание

Введение	3
Постановка задачи	4
Руководство пользователя	5
Руководство программиста	6
Описание структуры программы	6
Описание структур данных	6
Описание алгоритмов	6
Заключение	7

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
 - a. List.h
Содержит описание и реализацию шаблонного класса List
 - b. Node.h
Содержит описание и реализацию класса Node
 - c. Stack.h
Содержит описание и реализацию класса Stack
2. ../samples
 - a. 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 TKey>
class List {
private:
    Node<TKey>* root;
public:
    List();
    List(const List&);
    ~List();

    void Add(TKey);
    void Remove(TKey);
    void InsertBefore(TKey, TKey);
    void InsertAfter(TKey, TKey);
    void InsertEnd(TKey);
    void Print()const;
    int Size()const;
    TKey GetRoot()const;
    Node<TKey>* Search(TKey);
};
```

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 ключом;
5. InsertAfter - вставка после элемента с заданным ключом;
6. InsertEnd - вставка ключа в конец;
7. Print - вывод списка;
8. int Size() const - определение длины списка;

Шаблонный класс Node

```
template <typename TKey>
class Node {
public:
    TKey key;
    Node* 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<TKey>&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 "Stack Is Full";
    list->Add(key);
};
```

```

template <typename TKey>
TKey Stack<TKey>::Take()const
{
    if (IsEmpty())
        throw "Stack Is Empty";
    return list.GetRoot();
};

template <typename TKey>
TKey Stack<TKey>::Pop()
{
    if (IsEmpty())
        throw "Stack Is Empty";
    TKey key = list->GetRoot();
    try {
        list->Remove(key);
    }
    catch (const char* error) {
        cout << error << 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 стека на пустоту
5. IsFull – проверка стека 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 определенных 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 postfix::GetOperationPtr(char op) {
    int Ptr;
    switch (op) {
        case '*':
        case '/': Ptr = 3; break;
        case '+':
        case '-': Ptr = 2; break;
        case '(': Ptr = 1; break;
        case '=': Ptr = 0; break;
        default: Ptr = -1;
    }
    return Ptr;
}

int postfix::IsOperation(char op) {
    if (op == '+' || op == '-' || op == '*' || op == '/' || op == '=') return 1;
    else return 0;
}

bool postfix::Operand(const char Exp) {
    if ((Exp >= 65) && (Exp <= 90)) {
        return true;
    }
    else if (((Exp >= 40) && (Exp <= 43)) || (Exp == 45) || (Exp == 47) || (Exp ==
61)) {
        return false;
    }
    throw "Wrong operation " + Exp;
}

float postfix::calc_op(float one, float two, char op) {
    switch (op) {
        case '+':
            return (one + two);
        case '-':
            return (one - two);
        case '*':
            return (one * two);
        case '/':
            return (one / two);
        default:
            return -1;
    }
}

char* postfix::ConvertToPolish(char *InfixExp, int len) {
    char ch, t, *PolishExp = new char[strlen(InfixExp) + 1];
    int pos = 0;
    Stack<char> PolishStack, OperationStack;
    bool key;
    do {
        ch = InfixExp[pos++];
        if (isalpha(ch)) PolishStack.Push(ch);
        else if (ch == '(') OperationStack.Push(ch);
        else if (ch == ')') {
            while (1) {

```

```

        t = OperationStack.Pop();
        if (t == '(') break;
        PolishStack.Push(t);
    }
}
else if (IsOperation(ch)) {
    while (!OperationStack.IsEmpty()) {
        t = OperationStack.Pop();
        if (GetOperationPtr(ch) <= GetOperationPtr(t))
PolishStack.Push(t);
        else { OperationStack.Push(t); break; }
    }
    OperationStack.Push(ch);
}
} while ((ch != '=') && (pos < len));
pos = 0;
for (int i = 0; i < len; i++)
    if ((InfixExp[i] != '(') && (InfixExp[i] != ')')) pos++;
PolishExp[pos] = '\0';
PolishExp[--pos] = '=';
while (!PolishStack.IsEmpty()) PolishExp[--pos] = PolishStack.Pop();
return PolishExp;
}

float postfix::ConvertToInfix(char *Exp, int l) {
    Exp[l - 1] = '\0';
    l--;
    map<char, float> nums;
    for (size_t i = 0; i < l; i++) {
        if (postfix::Operand(Exp[i]) && nums.count(Exp[i]) == 0) {
            float var;
            cout << "Введите значение переменной: " <<
Exp[i] << " = ";
            cin >> var;
            nums.insert(pair<char, float>(Exp[i], var));
        }
    }
    Stack<float> end_list;
    for (int i = 0; i < l; i++) {
        char element = Exp[i];
        if (postfix::Operand(element)) {
            end_list.Push(nums[element]);
        }
        else {
            double two = end_list.Pop();
            double one = end_list.Pop();
            end_list.Push(calc_op(one, two, element));
        }
    }
    return end_list.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).