

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ
по лабораторной работе №2
по дисциплине «Алгоритмы и структуры данных»
Тема: « Рекурсия»

Студент гр. 8381

Преподаватель

Переверзев Д.Е.

Жангиров Т.Р.

Санкт-Петербург

2019

Цель работы

Ознакомиться с методами использования рекурсии и иерархического списка, написать программу преобразования выражения, хранящегося в иерархическом списке, в постфиксную форму с использованием рекурсии, а также реализовать проверку синтаксической корректности.

Теоретические положения

Рекурсия — вызов функции (процедуры) из неё же самой, непосредственно (*простая рекурсия*) или через другие функции (*сложная* или *косвенная рекурсия*), например, функция вызывает функцию, а функция — функцию. Количество вложенных вызовов функции или процедуры называется глубиной рекурсии. Рекурсивная программа позволяет описать повторяющееся или даже потенциально бесконечное вычисление, причём без явных повторений частей программы и использования циклов.

Иерархический список — список имеющий несколько уровней, в нашем случае, понижение уровня происходит при открытии скобки('('), соответственно при закрытии скобки происходит переход на предыдущий уровень.

Постфиксная запись представляет собой такую запись арифметического выражения, в которой сначала записываются операнды, а затем — знак операции. Например, для выражения $a + b * c$ постфиксная запись будет $a b c * +$. Здесь операндами операции $*$ будут b и c (два ближайших операнда), а операндами операции $+$ будут a и составной операнд $b c *$. Эта запись удобна тем, что она не требует скобок. Например, для выражения $(a + b) * c$ постфиксная запись будет $a b + c *$. В этой записи не требуется ставить скобки для того, чтобы изменить порядок вычисления, зависящий от приоритета операций, как в исходном выражении.

Задание

Вариант 19

Пусть выражение (логическое, арифметическое, алгебраическое*) представлено иерархическим списком. В выражение входят константы и переменные, которые являются атомами списка. Операции представляются в префиксной форме ((<операция> <аргументы>)), либо в постфиксной форме (<аргументы> <операция>). Аргументов может быть 1, 2 и более. Например (в префиксной форме): (+ a (* b (- c))) или (OR a (AND b (NOT c))).

В задании даётся один из следующих вариантов требуемого действия с выражением: *проверка синтаксической корректности, упрощение* (преобразование), *вычисление*.

Пример *упрощения*: (+ 0 (* 1 (+ a b))) преобразуется в (+ a b).

В задаче *вычисления* на входе дополнительно задаётся список значений переменных

((x₁ c₁) (x₂ c₂) ... (x_k c_k)),

где x_i – переменная, а c_i – её значение (константа).

В индивидуальном задании указывается: тип выражения (возможно дополнительно - состав операций), вариант действия и форма записи. Всего 9 заданий.

* - здесь примем такую терминологию: в *арифметическое* выражение входят операции +, -, *, /, а в *алгебраическое* – +, -, * и дополнительно некоторые функции.

19) арифметическое, проверка синтаксической корректности и деления на 0 (простая), постфиксная форма

Выполнение работы

1. Создание функции `prior()`, которая в зависимости от передаваемого символа возвращает его приоритет над остальными.
2. Функция `what()` принимает символ в качестве аргумента и возвращает число из диапазона $[-2;3]$ в зависимости от того, к какой группе принадлежит символ.
3. Создание функции, проверяющей исходные данные на синтаксическую корректность и деление на 0.
4. Функции `input()` производит ввод перевод введенного выражения в иерархический список.
5. Основная функция `fun()`, которая проходит по иерархическому списку и с помощью функции `postf()` переводит выражение в постфиксную запись по алгоритму:
 - I. Константы и переменные кладутся в формируемую запись в порядке их появления в исходном списке.
 - II. При появлении операции в исходном массиве:
 - (i) если в стеке нет операций или верхним элементом стека является открывающая скобка, операции кладётся в стек;
 - (ii) если новая операции имеет больший приоритет, чем верхняя операции в стеке, то новая операции кладётся в стек;
 - (iii) если новая операция имеет меньший или равный приоритет, чем верхняя операции в стеке, то операции, находящиеся в стеке, до ближайшей открывающей скобки или до операции с приоритетом меньшим, чем у новой операции, перекладываются в формируемую запись, а новая операции кладётся в стек.
 - III. Открывающая скобка кладётся в стек.
 - IV. Закрывающая скобка выталкивает из стека в формируемую запись все операции до ближайшей открывающей скобки, открывающая скобка удаляется из стека.
 - V. После того, как мы добрались до конца исходного выражения, операции, оставшиеся в стеке, перекладываются в формируемое выражение.

Тестирование программы

Входная строка	Строка после форматирования
$(a+b)*(c+d)-e$	$ab+cd+*e-$
$(a+b)+(c*d(g/m))-v/5+p-0-i$	$ab+cdgm/*v5/p0-+-+$
$(a+b)-c$	$ab+c-$
$(a*b**c)$	Содержит ошибку
$a-b*c-d$	$abc*-d-$

Выводы

В ходе лабораторной работы были изучены методы иерархического списка и метод перевода в постфиксную форму выражения хранящегося в иерархическом списке при помощи рекурсивных функций. А также реализованы этот методы на практике и выполнены проверки на синтаксическую корректность и деления на ноль.

Приложение А

Исходный код программы.

```
#include <iostream>
#include <cstdlib>
#include <fstream>

using namespace std;
ofstream out("./out.txt");

struct ELEM
{
    ELEM *next;
    bool event;
    union {
        ELEM *down; //true
        char simb; //false
    };
};

int what(char& simb)
{
    if((simb>='a'&&simb<='z')||(simb>='0'&&simb<='9'))
        return 3;
    else if(simb=='*'||simb=='-'||simb=='+'||simb=='/')
        return 2;
    else if(simb=='('||simb==')')
        return 1;
    else if(simb=='\n')
    {
        simb=0;
        return -1;
    }
    else if(simb>=32)
        return -2;
    return 0;
}

void push(ELEM *&head, char simb)
{
    ELEM *element = new ELEM;
    if (simb == '(')
    {
        head->next=NULL;
```

```

        head->down = element;
        head->event = true;
    }
    else if (simb != ')')
    {
        head->next = element;
        head->simb = simb;
        head->event = false;
    }
    else if (simb == ')')
    {
        head->next=element;
        element->simb=')';
        element->next=NULL;
        element->event=false;
    }
}

```

```

void input(ELEM *&head, string initial, int& position)
{
    if (position+1 == initial.size())
        return;
    position++;
    push(head,initial[position]);
    if (initial[position] == '(')
        input(head->down,initial,position);
    else if(initial[position+1]!='')
        input(head->next,initial,position);
    if(initial[position+1]=='')
    {
        if(head->event)
        {
            position+=1;
            push(head,initial[position]);
            input(head->next, initial, position);
        }
    }
    return;
}
}

```

```

void test(ELEM *&head, int len, int& position,string& help,string& finish,int& error)
{
    if (position >= len)
        return;
    position++;
    if(finish.find('(')!=string::npos)

```

```

{
    error+=1;
    return;
}
if(!(head->event))
    if(what(head->simb)==-2)
    {
        error+=1;
        out<<error<<". simbol: \"<<head->simb<<\"\\ (index of simbol: "<<position<<")\\n";
    }
if(!(head->event)&&head->simb<=32)
    return;
if (head->event)
{
    position++;
    if(what(head->down->simb)==2)
    {
        error+=1;
        out<<error<<". simbol: \"<<head->down->simb<<\"\\ (index of simbol: "<<position<<")
\\n";
    }
    test(head->down,len,position,help,finish,error);
}
if(head->next)
{
    if(!(head->next->event))
    {
        if(what(head->simb)==what(head->next->simb))
        {

            error+=1;
            out<<error<<". simbol: \"<<head->next->simb<<\"\\ (index of simbol: "<<position<<")
\\n";
        }
        if(head->simb=='/'&&head->next->simb=='0')
        {

            error+=1;
            out<<error<<". symbols:\"<<head->simb<<head->next->simb<<\"\\ (index of simbol:
"<<position<<")\\n";
        }
    }
    test(head->next,len,position,help,finish,error);
}
if(finish.find('')!=string::npos)
    error+=1;

```



```
}
```

```
int prior(char x)
```

```
{
```

```
    if ((x=='*')||(x=='/')) return 2;
```

```
    if ((x=='+')||(x=='-')) return 1;
```

```
    if ((x=='(')||(x=='))') return 0;
```

```
    return -1;
```

```
}
```

```
void postf(string& help,string& finish,char simb,int position)
```

```
{
```

```
    out<<"\n" <<position+1 <<'. ' <<simb <<"\t" <<help[help.size()-1] <<"\t";
```

```
    if(simb=='(')
```

```
    {
```

```
        help+=simb;
```

```
    }
```

```
    else if((simb=='+')||(simb=='-')||(simb=='/'))||(simb=='*'))
```

```
    {
```

```
        while((help.size()!=0)&&(prior(help[help.size()-1])>prior(simb)))
```

```
        {
```

```
            finish+=help[help.size()-1];
```

```
            help.resize(help.size()-1);
```

```
        out<<finish[finish.size()-1];
```

```
        }
```

```
        help+=simb;
```

```
    }
```

```
    else if(simb=='))')
```

```
    {
```

```
        while(help.size()&&help[help.size()-1]!='(')
```

```
        {
```

```
            finish+=help[help.size()-1];
```

```
            help.resize(help.size()-1);
```

```
        out<<finish[finish.size()-1];
```

```
        }
```

```
        help.resize(help.size()-1);
```

```
    }
```

```
    else
```

```
    {
```

```
        finish+=simb;
```

```
        out<<simb;
```

```
    }
```

```
}
```

```
void fun(ELEM *&head, int& len, int& position,string& help,string& finish)
```

```
{
```

```

if (position+1 == len)
    return;
position++;
if(!(head->event))
{
    postf(help,finish,head->simb,position);
}
if(!(head->event)&&head->simb<=' ')
{
    postf(help,finish,')',position);
    len++;
    return;
}
if (head->event)
{
    postf(help,finish,(' ',position);
    fun(head->down,len,position,help,finish);
    position++;
}
if(head->next)
{
    fun(head->next,len,position,help,finish);
}
}

```

```

int main()
{
    string initial;
    string finish,help;
    int error=0;
    ELEM *head = new ELEM;

    //in
    cout<<"Введите выражение: ";
    cin>>initial;
    out<<"Введенное выражение: "<<initial<<"\n\n";
    int len=initial.size();
    int position = -1;
    input(head, initial, position);

    //test
    out<<"Найденные ошибки:\n";
    position=-1;
    test(head,len,position,help,finish,error);
}

```

```
if(error)
{
    out<<"Найдено ошибок:"<<error<<endl;
    return 0;
}
else
{
    out<<"Ошибки отсутствуют\n";
}

//postf
position=-1;
fun(head,len,position,help,finish);

//out
for(int i=help.size()-1;i>=0;i--)
    finish+=help[i];
out<<endl<<"-----\nПостфиксная форма: "<<finish<<endl;
system("open ./out.txt");
return 0;
}
```