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 _____ **Переверзев Д.Е.**

Преподаватель _____ **Жангиров Т.Р.**

Санкт-Петербург

2019

Цель работы

Ознакомиться с методами использования рекурсии и написать программу преобразования выражения 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.

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.

Алгоритм перевода в постфиксную запись обрабатывает исходный массив лексем и строит новый массив из тех же лексем, расположенных в другом порядке. Кроме того, необходим еще стек – аналогичный массив, используемый для временного хранения операций.

Задание

Вариант 19

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

((x_1 c_1) (x_2 c_2) … (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 возвращай его приоритет над остальными.
2. Функция what() принимает символ в качестве аргумента и возвращает число из диапазона [0;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()/output() производят ввод и вывод соответсвенно.
5. Основная функция postf(), которая переводит выражение в постфиксную запись по алгоритму:

I. Константы и переменные кладутся в формируемую запись в порядке их появления в исходном массиве.

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)	ab+
(a+b)-c	ab+c-
(a*b*c)	abc**
a-b*c-d	abc*-d-

Выводы

В ходе лабораторной работы был изучен метод перевода в постфиксную форму при помощи рекурсивных функций. А также реализован этот метод и выполнена проверка на синтаксическую корректность и деления на ноль.

Приложение А

Исходный код программы.

```
#include<iostream>
#include <string>
#define ERROR "Error\n"
using namespace std;

int prior(char x)
{
   if ((x=='*')||(x=='/')) return 2;
   if ((x=='+')||(x=='-')) return 1;
   if ((x=='(')||(x==')')) return 0;
   return -1;
}

class stack
{
public :
   int top;
   string body;
   stack(){top=0;}
   bool empty(){return top==0;}
   char get_top_element(){return body[top];}
   int top_prior(){return prior(body[top]);}
   void push(char x)
   {
      top++;
      body[top]=x;
   }
   char pop()
   {
      top--;
      return body[top+1];
   }

};

int what(char simv)
{
  if(simv>='a'&&simv<='z')
            return 3;
  if(simv=='*'||simv=='-'||simv=='+'||simv=='/')
            return 2;
  if(simv=='('||simv==')')
```

```
            return 1;
  return 0;
}


void postf(string note,string &pnote,int &p, int &i,stack &s)
{


            cout<<"\n1."<<note[i]<<"\t"<<s.get_top_element()<<"\t";

      if (note[i]=='(') s.push(note[i]);
      else if ((note[i]=='+')||(note[i]=='-')||(note[i]=='/')||(note[i]=='*'))
      {
            //cout<<note[i]<<pnote[p]<<endl;

         while((!s.empty())&&(s.top_prior()>prior(note[i])))
         {
            p++;
            pnote[p]=s.pop();
            cout<<pnote[p];

         }
         s.push(note[i]);
      }
      else if(note[i]==')')
      {
            //cout<<note[i]<<pnote[p]<<endl;

         while((!s.empty())&&(s.get_top_element()!='('))
         {
            p++;
            pnote[p]=s.pop();
            cout<<pnote[p];
         }
         s.pop();
      }
      else
      {
            //cout<<note[i]<<pnote[p]<<endl;

         p++;
         pnote[p]=note[i];
         cout<<pnote[p];
      }
      if(i>=note.size())
```

```
            return;
    i++;
   postf(note,pnote,p,i,s);

   if(i>=note.size())
            while(!s.empty())
            {
               p++;
               pnote[p]=s.pop();
            }

    return;
}

 void test(string n,string p)
{
   int error=0;
   for(int i=0;i<n.size()-1;i++)
   {
            if(what(n[i])==0)
                     error+=1;
            if(n[i]=='('&&n[i+1]==')')
                     error+=1;
            if(what(n[i])==what(n[i+1])&&what(n[i])!=1)
                     error+=1;
            if(n[i]=='/'&&n[i+1]=='0')
                     error+=1;
   }
   if(p.find('(')!=-1||p.find(')')!=-1)
                     error+=1;

   if(error)
   {
            cout<<ERROR;
            exit(1);
   }
 }


void input(string& note)
{

   cin>>note;

}
```

```
void output(string pnote,int p)
{
  cout<<"Ошибок не обнаружено.\nПостфиксная запись: ";
  for(int i=1;i<=p;i++)
      cout<<pnote[i];
   cout<<endl;

}


int main(int argc, char* argv[])
{
   stack s;
   string note,pnote;
   int i=0,p=0;
   cout<<"Входны данные: ";
   if(argc<2) input(note);
   else
   {
            note=argv[1];
            cout<<note<<endl;
   }
  postf(note,pnote,p,i,s);
  test(note,pnote);
  output(pnote,p);

   return 0;
}
```