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

ОТЧЕТ

по лабораторной работе №3
по дисциплине «АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ»
Тема: ДЕРЕВЬЯ

Студент гр. 9303 _____ Халилов Ш.А.

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург
2020

Цель работы.

Реализовать алгоритм дерева и леса для преобразования формулы из инфиксной в префиксную форму

Задание.

С помощью построения дерева-формулы t преобразовать заданную формулу f из инфиксной формы в префиксную (перечисление узлов t в порядке КЛП) и в постфиксную (перечисление в порядке ЛПК); - если в дереве-формуле t терминалами являются только цифры, то вычислить (как целое число) значение дерева-формулы t.

Основные теоретические положения.

Дерево – конечное множество T, состоящее из одного или более узлов, таких, что

а) имеется один специально обозначенный узел, называемый корнем данного дерева;

б) остальные узлы (исключая корень) содержатся в m ◌ 0 попарно не пересекающихся множествах T1, T2, ..., Tm, каждое из которых, в свою очередь, является деревом. Деревья T1, T2, ..., Tm называются под деревьями данного дерева.

При программировании и разработке вычислительных алгоритмов удобно использовать именно такое рекурсивное определение, поскольку рекурсивность является естественной характеристикой этой структуры данных.

Каждый узел дерева является корнем некоторого поддерева. В том случае, когда множество поддеревьев такого корня пусто, этот узел называется концевым узлом, или листом. Уровень узла определяется рекурсивно следующим образом: 1) корень имеет уровень 1; 2) другие узлы имеют уровень, на единицу больший их уровня в содержащем их поддереве этого корня.

Говорят, что каждый корень является отцом корней своих поддеревьев и что последние являются сыновьями своего отца и братьями между собой.

Говорят также, что узел n – предок узла m (а узел m – потомок узла n), если n – либо отец m, либо отец некоторого предка m. Если в определении дерева существен порядок перечисления поддеревьев T1, T2, ..., Tm, то дерево называют упорядоченным и говорят о «первом» (T1), «втором» (T2) и т. д. поддеревьях данного корня. Далее будем считать, что все рассматриваемые деревья являются упорядоченными, если явно не оговорено противное. Отметим также, что в терминологии теории графов определенное ранее упорядоченное дерево более полно называлось бы «конечным ориентированным (корневым) упорядоченным деревом».

Лес – это множество (обычно упорядоченное), состоящее из некоторого (быть может, равного нулю) числа непересекающихся деревьев. Используя понятие леса, пункт б в определении дерева можно было бы сформулировать так: узлы дерева, за исключением корня, образуют лес.

Выполнение работы.

Для реализации алгоритм дерева и леса для преобразования формулы из инфиксной в префиксную форму были реализованы следующие функции:

class Btree — класс бинарное дерева

checkSymbol — функция для определение символа.

getLeft - функция для получения левой части строки.

getRight- функция для получения правой части строки.

ToPrefix — функция получает строка ссылки на объекта класса Btree и индекс, функция ищет первый оператор и если найдет то по этом индексом разделяет строку на две части и с левой части вызывает повторно эту функцию для левого узла объекта класса Btree, с правой части вызывается для правого узла.

ShowTree — функция позволяет получить нижний результат пробегая по всем лесам по принципу поиска в глубину. И выводит в нужном порядке

int getResult(BTree &head) - эта функция вычитает выражения используя принцип поиска в глубину

Пример работы.

Примеры работ есть в файле infixToPrefix.txt

Выводы.

При реализация алгоритмов дерева и леса для преобразования формулы из инфиксной в префиксную форму, были изучен основные принципы работы с бинарным древом.

Приложения А

```
#include <iostream>
#include <fstream>
#include <cstring>

#define PATERN "+-* ()0123456789slqrtabcdefghijkmnopyuvxzw"

using namespace std;

class BTree
{
public:
    string date;
    BTree *lSub;
    BTree *rSub;
    bool isleft(){
        if( lSub )
            return true;
        return false;
    }
    bool isRight(){
        if( rSub )
            return true;
        return false;
    }
};


int checkSymbol( char c ){
    int n;
    for( n = 0; c != PATERN[n]; n++);
    return n;
}
```

```
string getLeft( string &str, int to)
{
    string out = "";
    int ccc = true;
    for (int i = 0; i < to; i++)
    {
        if(str[i] == '(' && ccc )
        {
            ccc = false;
            continue;
        }
        out += str[i];
    }
    return out;
}

string getRight( string &str, int from)
{
    string out = "";
    for (int i = from+1; i < str.length()-1; i++)
    {
        out += str[i];
    }
    return out;
}
void toPrefix(string str, BTree &head ){

    int c = -1, i = 0;
    int countC = 0;
    while (i < str.length())
    {
        if(str[i] == '(') countC++;
        if(str[i] == ')') countC--;

        if ( checkSymbol(str[i]) <= 3 && countC == 1) {
```

```
            c  = checkSymbol(str[i]);
            break;
        }
        i++;
    }
    if(c >= 0 && c < 3 ) {

         switch (c)
        {
            case 0:
                head.date = "+";
                break;
            case 1:
                head.date = "-";
                break;
            default:
                head.date = "*";
                break;
        }
        head.lSub = new BTree;
        string L = getLeft(str, i);
        cout << "tmp L: " + L +"\n";
        toPrefix(L, *(head.lSub)) ;

        head.rSub = new BTree;
        string R = getRight(str, i);
        cout << "tmp R: " + R +"\n";
        toPrefix(R, *(head.rSub));
    }
    else
    {
        head.date = str;
    }
}
string ShowTree(BTree &head, ofstream &fout, int n){
```

```
    cout << head.date;

    for (int i = 0; i < n; i++) {

        fout << "   ";
        cout << "   ";
    }
    if(head.date == "+" || head.date == "-" || head.date == "*" ||
head.date == "/")
    {
        string oL ="";
        string oR ="";
        cout << head.date << "\n";
        fout << head.date << "\n";
        return head.date
                + " "
                + ShowTree( *(head.lSub), fout, n+1 )
                + " "
                + ShowTree( *(head.rSub), fout, n+1 );
    }

    cout << head.date << "\n";
    fout << head.date << "\n";
    return head.date;
}

int getResult( BTree &head, bool &ch )
{
    if( head.isRight() || head.isleft())
    {
        int left = getResult( *(head.lSub ) , ch),
            right = getResult( *(head.rSub ), ch);

        return head.date == "+"
                ? left + right
                : head.date == "-"
```

```
                    ? left - right
                    : left * right;

    }
    int out = 0, i = 0;
    while ( i < head.date.length())
    {
        int tmp = checkSymbol(head.date[i]);
        if(tmp > 9) ch = false;
        out = out*10 + tmp;
        i++;
    }
    return out;
}
void cleanSpace( string& str ){
    string tmp = "";
    for (int i = 0; i < str.length(); i++)
    {
        if(str[i] == ' ') continue;
        tmp += str[i];
    }
    str = tmp;
}
int main(int argc, char const *argv[]){

    string str;
    ifstream fin;
    ofstream fout;

    char c[240];
    if( argc > 1 ){

        fin.open(argv[1]);
        if( argc > 2 ) {
            fout.open(argv[2], ios_base::app);
        }
```

```
    else {
        fout.open("infixToPrefix.txt", ios_base::app);
    }

    while (!fin.eof())
    {
        fin.getline(c, 240);
        str = c;
        BTree head;
        cleanSpace(str);
        fout << "\n[::new::]\n" << str << "\n";
        toPrefix( str, head);
        fout << "\n" << ShowTree( head, fout, 1) << "\n";

        bool isOnlyNum = true;
        int result = getResult( head, isOnlyNum );
        if( isOnlyNum )
            fout <<  result << "\n";
    }
    fin.close();
    fout.close();
}
else {

    fin.getline(c, 240);
    str = c;
    BTree head;
    toPrefix( str, head);

    fout.open("infixToPrefix.txt", ios_base::app);
    cout << str << "\n";
    fout << "\n\n[ IN ]: \n" << str;
    fout <<"\n[ Infix out ]: \n" + ShowTree(head, fout, 1);
    fout.close();
}
return 0;
```

```
}
```