АКМИНОБРНАУКИ 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: Программирование рекурсивных алгоритмов

Студент гр. 9303 _____ Халилов Ш.А.

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург
2020

Цель работы.

Реализовать алгоритм для проверки синтаксической корректности выражении на префиксной форме

Задание.

Алгебраическое (+, -, *, sqrt(), log(,)), проверка синтаксической корректности, простая проверка log(,), префиксная форма.

Основные теоретические положения.

Рекурсивный - это объект, который содержит сам себя или определяется с помощью самого себя. Сила рекурсии заключается в том, что она позволяет вам определять бесконечное количество объектов с помощью конечного оператора. Точно так же бесконечные вычисления можно описать с помощью конечной рекурсивной программы. Рекурсивные алгоритмы лучше всего использовать, когда решаемая проблема, вычисляемая функция или обрабатываемая структура данных определяются с помощью рекурсии. Если процедура (функция) P содержит явный вызов самой себя, она называется непосредственно рекурсивной. Если P содержит вызов процедуры (функции) Q, которая содержит (прямо или косвенно) вызов P, то P называется косвенно рекурсивным. Многие известные функции можно определить рекурсивно. Например, факториал, который присутствует почти во всех учебниках программирования, а также наибольший общий делитель, числа Фибоначчи, степенная функция и т. Д.

Двои́чное де́рево — иерархическая структура данных, в которой каждый узел имеет не более двух потомков (детей). Как правило, первый называется родительским узлом, а дети называются левым и правым наследниками. Двоичное дерево не является упорядоченным ориентированным деревом.

Выполнение работы.

Реализовать алгоритм для проверки синтаксической корректности выражении на префиксной форме были реализованы следующие функции:

Btree - это двоичная древовидная структура, она имеет 3 поля это

string data - поля для данных

Btree * Left, * Right - указатель на потомков

Дальнейшем вся алгоритм строится над этой структурой.

checkSymbol - функция для проверки символа. Она принимает символ и возвращает индекс символа из строки, которой объявление в макросе

bool Sqrt и Log - функции для проверки правильности корней и логарифмов, они используют регулярное выражение.

CreatForest - Эта функция получает строку и строит лес из двоичных деревьев, в функцией проверяется символы строки, если символ является пробелом, тогда функция игнорирует их. если символ оказывается оператором, то создаются левый и правый узлы и для них повторно вызывается функция в остальном случае символы будут приняты как операнды

checkPrefix - функция проверки корректности префиксного выражения. функция будет пробегать весь лес по принципу поиска глубины и, если есть оператор без операндов, вернет false

Пример работи.

Пример работы находится в файле result.txt

Выводы.

При выполнения работы были изучены принципы работы с бинарными деревами и лес. И была создана программа для проверки корректности префиксного выражения.

Приложения А

```
#include <iostream>
#include <fstream>
#include <regex>

using namespace std;

#define PATERN_LOG "log\\s?\\(\\s?\\d+\\s?,\\s?\\d+\\s?\\)"
#define PATERN_SQRT "sqrt\\s?\\(\\s?\\d+\\s?\\)"
#define PATERN "0123456789+-*/ ()slqrtabcdefghijkmnopyuvxzw"
#define PLUS "+";
#define MINUS "-";
#define MULTIPLICATION "*";
#define DIVISION "/";
typedef struct BTree
{
    string date;
    BTree *Left;
    BTree *Right;
} BTree;
int checkSymbol( char & c ){
    int n;
    for( n = 0; c != PATERN[n]; n++);
    return n;
}

bool Sqrt( string &str, string &out_str, int &Index) {
    string tmp = "";
    int i;
    for (i = Index; str[i] != ')'; i++){
        tmp += str[i];
    }
    tmp += str[i];
    if( regex_match(tmp, regex(PATERN_SQRT))) {
        out_str = tmp;
```

```
        Index = i+1;
        return true;
    }
    return false;
}
bool Log( string &str, string &out_str, int &Index) {
    string tmp = "";
    int i;
    for (i = Index; str[i] != ')'; i++){
        tmp += str[i];
    }
    tmp += str[i];

    if( regex_match(tmp, regex(PATERN_LOG))) {

        out_str = tmp;
        Index = i+1;
        return true;
    }
    return false;
}
void creatForest(string &str, int &index, BTree &Head) {

    int num = 0, c;
    string tmp_str = "";
    c = checkSymbol(str[index]);
    if(c == 14) {
        while(c == 14 ) {
            index++;
            c = checkSymbol(str[index]);
        }
    }
    if( c >= 10 && c <= 13 ) {
        index++;
        switch (c){
            case 10:
```

```
            Head.date = PLUS;
            break;
        case 11:
            Head.date = MINUS;
            break;
        case 12:
            Head.date = MULTIPLICATION;
            break;
        default:
            Head.date = DIVISION;
            break;
    }
    Head.Left = new BTree;
    creatForest ( str, index, *(Head.Left) );
    Head.Right = new BTree;
    creatForest ( str, index, *(Head.Right));
}
else if( c >= 0 && c <= 9 ) {
    for (;;index++) {
        c = checkSymbol(str[index]);
        if( c > 9 || c < 0 ) break;
        tmp_str += str[index];
    }
    Head.date = tmp_str;
}
else if( c == 15 ) {
    for (;;index++) {

        tmp_str += str[index];
        if( c == 16 ) break;
        c = checkSymbol(str[index]);
    }
    Head.date = tmp_str;
}
else if( c == 17) {
    if( !Sqrt( str, tmp_str, index)) {
```

```
            Head.date = "";
            return;
        }
        Head.date = tmp_str;
    }
    else if( c == 18) {
        if(!Log( str, tmp_str, index)) {
            Head.date = "";
            return;
        }
        Head.date = tmp_str;
    }
    else if( c >= 19 && c < 43) {
        Head.date = str[index++];
    }
}


bool checkPrefix(BTree &head){

    if( head.date == "+" || head.date == "-" || head.date == "*"
|| head.date == "/" )
    {
        return checkPrefix( *(head.Left)) &&
checkPrefix( *(head.Right));
    }
    else if( head.date.empty())
        return false;
    return true;
}
string ShowTree(BTree &head, ofstream &fout, int n){
    for (int i = 0; i < n; i++) {
        fout << "   ";
        cout << "   ";
    }
    string out = "";
```

```
    if(head.date == "+" || head.date == "-" || head.date == "*" ||
head.date == "/")
    {
        cout << head.date << "\n";
        fout << head.date << "\n";

        out += "(" + ShowTree( *(head.Left), fout, n+1 );
        return out + head.date + ShowTree( *(head.Right), fout,
n+1)+")";
    }

    cout << head.date << "\n";
    fout << head.date << "\n";
    return head.date;
}
int main(int argc, char const *argv[]){

    string str;
    ifstream fin;
    ofstream fout;
    int index;
    char c[240];
    if( argc > 1 ){
        fin.open(argv[1]);
        if( argc > 2 ) {
            fout.open(argv[2], ios_base::app);
        }
        else {
            fout.open("res.txt", ios_base::app);
        }
        while (!fin.eof())
        {
            fin.getline(c, 240);
            str = c;
            index = 0;
```

```
            BTree head;

            creatForest( str, index, head);

            cout << str << "\n";
            fout << "\n\n[ IN ]: \n" << str;
            if( checkPrefix(head) ){
                cout <<"[ Prefix ] \n";
                fout <<"\t[ Prefix ] \n ";
            }
            else {
                cout <<"[ Not Prefix ] \n";
                fout <<"\t[ Not Prefix ] \n";
            }
            fout <<"\n[ Infix out ]: \n" + ShowTree(head, fout,
1);

        }
        fin.close();
        fout.close();
    }
    else {

        fin.getline(c, 240);
        str = c;
        BTree head;
        index = 0;
        creatForest( str, index, head);
        //  ofstream fout;
        fout.open("result.txt", ios_base::app);

        cout << str << "\n";
        fout << "\n\n[ IN ]: \n" << str;
        if( checkPrefix(head) ){
            cout <<"[ Prefix ] \n";
            fout <<"\t[ Prefix ] \n ";
```

```
        }
        else {
            cout <<"[ Not Prefix ] \n";
            fout <<"\t[ Not Prefix ] \n";
        }
        fout <<"\n[ Infix out ]: \n" + ShowTree(head, fout, 1);
        fout.close();
    }
    return 0;
}
```