

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

ОТЧЕТ  
по лабораторной работе №4  
по дисциплине «Алгоритмы и структуры данных»  
Тема: Деревья

Студент гр. 7383

\_\_\_\_\_

Лосев М. Л.

Преподаватель

\_\_\_\_\_

Размочаева Н. В.

Санкт-Петербург

2018

## СОДЕРЖАНИЕ

1. ЦЕЛЬ РАБОТЫ.....	3
2. РЕАЛИЗАЦИЯ ЗАДАЧИ.....	4
3. ТЕСТИРОВАНИЕ.....	5
4 . ВЫВОД.....	5
ПРИЛОЖЕНИЕ А. Исходный Код программы.....	6
ПРИЛОЖЕНИЕ Б. Тестовые случаи.....	14

## 1. ЦЕЛЬ РАБОТЫ

Цель работы: Познакомиться со структурой данных – деревом, освоить на практике использование деревьев для решения задач.

Формулировка задачи: Формулу вида

$\langle \text{формула} \rangle ::= \langle \text{терминал} \rangle \mid ( \langle \text{формула} \rangle \langle \text{знак} \rangle \langle \text{формула} \rangle )$

$\langle \text{знак} \rangle ::= + \mid - \mid *$

$\langle \text{терминал} \rangle ::= 0 \mid 1 \mid \dots \mid 9 \mid a \mid b \mid \dots \mid z$

можно представить в виде бинарного дерева («дерева-формулы») с элементами типа *char* согласно следующим правилам:

- формула из одного терминала представляется деревом из одной вершины с этим терминалом;

- формула вида  $(f_1 \ s \ f_2)$  представляется деревом, в котором корень - это знак  $s$ , а левое и правое поддеревья - соответствующие представления формул  $f_1$  и  $f_2$ . Например, формула  $(5 * (a + 3))$  представляется деревом-формулой, показанной на рис. 1.

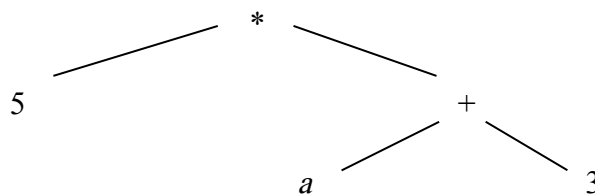


Рис. 1. Дерево-формула

Требуется:

- для заданной формулы  $f$  построить дерево-формулу  $t$ ;
- для заданного дерева-формулы  $t$  напечатать соответствующую формулу  $f$ ;
- с помощью построения дерева-формулы  $t$  преобразовать заданную формулу  $f$  из инфиксной формы в префиксную (перечисление узлов  $t$  в порядке КЛП) или в постфиксную (перечисление в порядке ЛПК);
- преобразовать дерево-формулу  $t$ , заменяя в нем все поддеревья, соответствующие формулам  $((f_1 * f_2) + (f_1 * f_3))$  и  $((f_1 * f_3) + (f_2 * f_3))$ , на поддеревья, соответствующие формулам  $(f_1 * (f_2 + f_3))$  и  $((f_1 + f_2) * f_3)$ ;

## 2. РЕАЛИЗАЦИЯ ЗАДАЧИ

### 2.1 ИСПОЛЬЗУЕМЫЕ СТРУКТУРЫ ДАННЫХ

*node* – класс, который представляет из себя узел бинарного дерева в ссылочной реализации. Поля структуры *node*:

- *info* – значение узла, имеет тип *base* (определен как *char*).
- *lt* – указатель на левого сына.
- *rt* – указатель на правого сына.

Методы класса *node*:

- Конструктор *node* – обнуляет указатели на детей.

### 2.2 ИСПОЛЬЗУЕМЫЕ ФУНКЦИИ

*Create* – возвращает пустое бинарное дерево.

*isNull* – принимает узел бинарного дерева, возвращает *true*, если узел пустой, и *false*, если нет.

*RootBT* – принимает узел дерева, возвращает значение этого узла.

*Left* – принимает узел дерева, возвращает левого сына.

*Right* – принимает узел дерева, возвращает правого сына.

*ConsBT* – принимает значение типа *base* и два узла дерева. Создает новый узел с полученным значением, для которого полученные узлы являются детьми, и возвращает этот узел.

*destroy* – принимает дерево, очищает память, занятую им. Не возвращает ничего.

*isEqual* – принимает два узла, возвращает *true*, если деревья, корнями которых являются принятые узлы, равны, и *false*, если нет.

*isTerminal* – принимает символ и возвращает *true*, если он является терминалом, или *false*, если нет.

*isSign* – принимает символ и возвращает *true*, если он является знаком, или *false*, если нет.

*MakeNode* – принимает символ, создает бездетный узел дерева, значение которого равно этому символу, и возвращает созданный узел.

PrintBt – принимает бинарное дерево и выводит его скобочную запись.

SkipSpaces – принимает поток ввода и считывает оттуда все пробелы, идущие подряд.

TreeToForm – принимает дерево-формулу, создает строку с соответствующей этому дереву формулой, и возвращает созданную строку.

FormToTree – принимает поток ввода, считывает оттуда формулу, строит по ней дерево-формулу и возвращает построенное дерево.

Transform – принимает узел дерева и выполняет над ним преобразования, описанные в пункте «з».

ProcTree – принимает дерево-формулу и выполняет над всеми его узлами, если это возможно, преобразования, описанные в пункте «з».

Translate – принимает дерево-формулу, создает строку, содержащую префиксную запись формулы, соответствующей полученному дереву, и возвращает созданную строку (префиксную запись формулы).

print – получает дерево, делает его отрисовку.

### **3. ТЕСТИРОВАНИЕ**

#### **3.1 ПРОЦЕСС ТЕСТИРОВАНИЯ**

Программа собрана в операционной системе Ubuntu 18.04 компилятором gcc. В других ОС тестирование не проводилось.

#### **3.1 РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ**

Тестовые случаи представлены в приложении Б. Ошибки выявить не удалось. По результатам тестирования было показано, что задача выполнена.

### **4 . ВЫВОД**

В ходе работы была написана программа на языке C++, которая строит дерево-формулу, соответствующую формуле, выполняет на нем преобразования, преобразует формулу в инфиксной форме в формулу в префиксной с помощью дерева и отрисовывает дерево. Был получен опыт в использовании бинарных деревьев в ссылочной реализации.

## ПРИЛОЖЕНИЕ А

### ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл MakeFile:

```
all: work_bt.o bt_implementation.o functions.o
    g++ work_bt.o  bt_implementation.o functions.o -o work_bt

work_bt.o: work_bt.cpp Btree.h functions.h
    g++ -c work_bt.cpp
functions.o: functions.cpp functions.h Btree.h
    g++ -c functions.cpp

bt_implementation.o: bt_implementation.cpp Btree.h
    g++ -c bt_implementation.cpp
clean:
    rm *.o bt_implementation
```

Файл Btree.h:

```
#pragma once

namespace binTree_modul
{
//-----
    typedef char base;

    struct node {
        base info;
        node *lt;
        node *rt;
        // constructor
        node ()
        {
            lt = NULL; rt = NULL;
        }
    };

    typedef node *binTree; // "представитель" бинарного дерева

    binTree Create(void);
    bool isNull(binTree);
    base RootBT (binTree); // для непустого бин.дерева
    binTree Left (binTree); // для непустого бин.дерева
    binTree Right (binTree); // для непустого бин.дерева
    binTree ConsBT(const base x, binTree lst, binTree rst);
    void destroy (binTree&);
    bool isEqual(binTree a, binTree b);

} // end of namespace binTree_modul
```

Файл functions.h:

```
#pragma once
#include <iostream>
#include "Btree.h"
```

```

using namespace std;
using namespace binTree_modul;

namespace Losev_Mikhail_lab4_namespace
{
    bool isTerminal( const char c );
    bool isSign( const char c );
    binTree MakeNode( base c );
    void PrintBt(binTree b);
    void SkipSpaces(istream &in);
    string TreeToForm(binTree b);
    binTree FormToTree(istream &in);
    void Transform(binTree b);
    void ProcTree(binTree b);
    string Translate(binTree b);
    void print(binTree f, int l);
}

```

Файл `bt_implementation.cpp`:

```

/ Implementation - Реализация АД "Бинарное дерево" (в процедурно-модульной
парадигме)
#include <iostream>
#include <cstdlib>
#include "Btree.h"
using namespace std ;

namespace binTree_modul
{
    //-----
    binTree Create()
    {
        return NULL;
    }
    //-----
    bool isNull(binTree b)
    {
        return (b == NULL);
    }
    //-----
    base RootBT (binTree b) // для непустого бин.дерева
    {
        if (b == NULL) { cerr << "Error: RootBT(null) \n"; exit(1); }
        else return b->info;
    }
    //-----
    binTree Left (binTree b) // для непустого бин.дерева
    {
        if (b == NULL) { cerr << "Error: Left(null) \n"; exit(1); }
        else return b ->lt;
    }
    //-----
    binTree Right (binTree b) // для непустого бин.дерева
    {
        if (b == NULL) { cerr << "Error: Right(null) \n"; exit(1); }
    }
}

```

```

        else return b->rt;
    }
//-----
binTree ConsBT(const base x, binTree lst, binTree rst)
{
    binTree p;
    p = new node;
    if ( p != NULL) {
        p ->info = x;
        p ->lt = lst;
        p ->rt = rst;
        return p;
    }
    else {cerr << "Memory not enough\n"; exit(1);}
}

bool isEqual(binTree a, binTree b)
{
    if (a == NULL && b == NULL)
        return true;
    if (a == NULL || b == NULL)
        return false;
    return isEqual(a->lt, b->lt) && isEqual(a->rt, b->rt) && a->info == b-
>info;
}

//-----
void destroy (binTree &b)
{
    if (b != NULL) {
        destroy (b->lt);
        destroy (b->rt);
        delete b;
        b = NULL;
    }
}

} // end of namespace

```

Файл functions.cpp:

```

#include <iostream>
#include <fstream>
#include <fstream>
#include <cstdlib>
#include <cstring>
#include "Btree.h"

using namespace std;
using namespace binTree_modul;

// вывод дерева как дерева
// более информативные ошибки

namespace Losev_Mikhail_lab4_namespace
{
    bool isTerminal( const char c )
    {
        return (c >= 'a' && c <= 'z') || (c >= '0' && c <= '9');
    }

    bool isSign( const char c )

```



```

{
    return c == '-' || c == '+' || c == '*';
}

binTree MakeLeaf( base c )
{
    return ConsBT(c, nullptr, nullptr);
}

void PrintBt(binTree b)
{
    if (b == NULL)
        return;

    if (b->lt == NULL && b->rt == NULL){
        cout << ' ' << b->info << ' ';
        return;
    }

    cout << "( ";
    PrintBt(b->lt);
    cout << ' ' << b->info << ' ';
    PrintBt(b->rt);
    cout << ")";
}

void SkipSpaces(istream &in)
{
    base c;
    do
    {
        c = in.peek();
        if (c == ' ')
            c = in.get();
    } while (c == ' ');
}

string TreeToForm(binTree b)
{
    string str = "";
    if (b == NULL)
        return str;

    if (b->lt == NULL && b->rt == NULL){
        str += ' ';
        str += b->info;
        str += ' ';
        return str;
    }

    str += "( ";
    str += TreeToForm(b->lt);
    str += ' ';
    str += b->info;
    str += ' ';
    str += TreeToForm(b->rt);
    str += ")";
}

binTree FormToTree(istream &in)

```

```

{
    base c, sign;
    binTree left, right;

    SkipSpaces(in);
    c = in.get();

    if (isTerminal(c)) // если формула является терминалом
        return MakeLeaf(c);

    if (c == '(') { // если формула имеет вид (<терм><знак><терм>)
        SkipSpaces(in);
        left = FormToTree(in); // первая формула

        SkipSpaces(in);
        c = in.get();
        if (isSign(c))
            sign = c; // знак
        else { cerr << "error: sign expected" << endl; return NULL; }
        // если после первой формулы нет знака

        SkipSpaces(in);
        right = FormToTree(in); // вторая формула

        SkipSpaces(in); // проверка скобок
        c = in.get();
        if (c != ')') { cerr << "error: ')' expected" << endl; return
NULL; }

        // формула вида (<терм><знак><терм>) заканчивается скобкой

        left = ConsBT(sign, left, right); // делаем из знака и двух
формул формулу
        return left;
    }
    else {
        if (isSign(c)) cerr << "error: extra sign in formula" << endl;
        else cerr << "error: external symbol if formula" << endl;
        return FormToTree(in);
        //return NULL;
    } // формула является либо термом либо имеет вид (<терм><знак><терм>)
и должна начинаться скобкой
}

```

```

void Transform(binTree b)
{
    // ((f1 * f2) + (f1 * f3)) -> (f1 * (f2 + f3))
    if (isEqual(b->lt->lt, b->rt->lt)){
        binTree f1, f2, f3;

        f1 = b->lt->lt;
        f2 = b->lt->rt;
        f3 = b->rt->rt;

        b->info = '*';
        b->lt = f1;
        b->rt->info = '+';
        b->rt->lt = f2;
        b->rt->rt = f3;
    }
}

```

```

        return;
    }
    // ((f1 * f2) + (f3 * f1)) -> (f1 * (f2 + f3))
    if (isEqual(b->lt->lt, b->rt->rt)){
        binTree f1, f2, f3;

        f1 = b->lt->lt;
        f2 = b->lt->rt;
        f3 = b->rt->lt;

        b->info = '*';
        b->lt = f1;
        b->rt->info = '+';
        b->rt->lt = f2;
        b->rt->rt = f3;

        return;
    }
    // ((f2 * f1) + (f1 * f3)) -> (f1 * (f2 + f3))
    if (isEqual(b->lt->rt, b->rt->lt)){
        binTree f1, f2, f3;

        f1 = b->lt->rt;
        f2 = b->lt->lt;
        f3 = b->rt->rt;

        b->info = '*';
        b->lt = f1;
        b->rt->info = '+';
        b->rt->lt = f2;
        b->rt->rt = f3;

        return;
    }
    // ((f2 * f1) + (f3 * f1)) -> (f1 * (f2 + f3))
    if (isEqual(b->lt->rt, b->rt->rt)){
        binTree f1, f2, f3;

        f1 = b->lt->rt;
        f2 = b->lt->lt;
        f3 = b->rt->lt;

        b->info = '*';
        b->lt = f1;
        b->rt->info = '+';
        b->rt->lt = f2;
        b->rt->rt = f3;

        return;
    }
}

void ProcTree(binTree b)
{
    if (b == NULL)
        return;
    ProcTree(b->lt);

```

```

ProcTree(b->rt);

if (b == NULL ||
    b->lt == NULL || b->lt->lt == NULL || b->lt->rt == NULL ||
    b->rt == NULL || b->rt->lt == NULL || b->rt->rt == NULL)
    return; // проверяем достаточное количество потомков узла

if (b->lt->info != '*' || b->rt->info != '*' || b->info != '+')
    return; // проверяем знаки

// не проверяем, что есть множитель, который можно вынести
// Transform сама проверит это
Transform(b);

}

string Translate(binTree b)
{
    string str = "";

    if (b == NULL)
        return str;
    str += ' ';
    str += b->info;
    str += ' ';
    if (b->lt != NULL){
        str += ' ';
        str += Translate(b->lt);
        str += ' ';
    }
    if (b->rt != NULL){
        str += ' ';
        str += Translate(b->rt);
        str += ' ';
    }
}

void print(binTree f, int l)
{
    if(f == nullptr){
        for(int i = 0; i < l; i++)
            cout << "\t";
        cout << '#' << endl;
        return;
    }

    print(f->rt, l+1);

    for(int i = 0; i < l; i++)
        cout << "\t";

    cout << f->info << endl;

    print(f->lt, l+1);
}
}

```

Файл work\_bt.cpp:

```

#include <iostream>
#include <fstream>
#include <fstream>
#include <cstdlib>
#include <cstring>
#include "Btree.h"
#include "functions.h"

using namespace std ;
using namespace binTree_modul;
using namespace Losev_Mikhail_lab4_namespace;

int main ()
{
    string formula;
    binTree b;

    cout << "Водите формулу: ";
    b = FormToTree(cin);
    cout << "Дерево-формула построено:" << endl;
    formula = TreeToForm(b);
    cout << formula << endl;

    cout << "Преобразование дерева-формулы..." << endl;
    ProcTree(b);
    formula = TreeToForm(b);
    cout << formula << endl;

    cout << "Перевод дерева-формулы в префиксную форму..." << endl;
    formula = Translate(b);
    cout << formula << endl;

    cout << "Визуализация дерева-формулы: " << endl;
    print(b, 0);

    destroy (b);
    cout << endl;
    return (0);
}

```

## ПРИЛОЖЕНИЕ Б ТЕСТОВЫЕ СЛУЧАИ

Входное выражение	Вывод программы	Корректность выполнения
$(a + b)$	Дерево-формула построено: $( a + b )$ Преобразование дерева-формулы... $( a + b )$ Перевод дерева-формулы в префиксную форму... $+ a b$	да
$(a ++ b)$	error: extra sign in formula Дерево-формула построено: $( a + b )$ Преобразование дерева-формулы... $( a + b )$ Перевод дерева-формулы в префиксную форму... $+ a b$ Визуализация дерева-формулы: <pre>       #     b       # +       #     a       # </pre>	да
$(a + b$	error: ')' expected Дерево-формула построено:  Преобразование дерева-формулы...  Перевод дерева-формулы в префиксную форму...  Визуализация дерева-формулы:	да
$((a+b)*c)+(3*(a+b))$	Дерево-формула построено: $(( ( a + b ) * c ) + ( 3 * ( a + b ) ) )$ Преобразование дерева-формулы... $(( a + b ) * ( c + 3 ) )$ Перевод дерева-формулы в префиксную форму... $* + a b + c 3$ Визуализация дерева-формулы: <pre>       #     3       # +       #     c       # </pre>	да

	<pre> *       #     b       #   +       #     a       # </pre>	
a+b	<p>Дерево-формула построено:</p> <pre> a Преобразование дерева-формулы... a Перевод дерева-формулы в префиксную форму... a Визуализация дерева-формулы:   # a   # </pre>	да