

**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

\_\_\_\_\_

Корякин М.П.

Преподаватель

\_\_\_\_\_

Размочаева Н. В.

Санкт-Петербург

2018

## СОДЕРЖАНИЕ

1. ЦЕЛЬ РАБОТЫ.....	3
2. РЕАЛИЗАЦИЯ ЗАДАЧИ .....	4
3. ТЕСТИРОВАНИЕ .....	5
4 . ВЫВОД.....	6
5.ПРИЛОЖЕНИЕ А .....	7

## 1. ЦЕЛЬ РАБОТЫ

Цель работы: Познакомиться со структурой данных – деревом, освоить на практике использование деревьев для решения задач. Вариант 9(абвж)-д.

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.

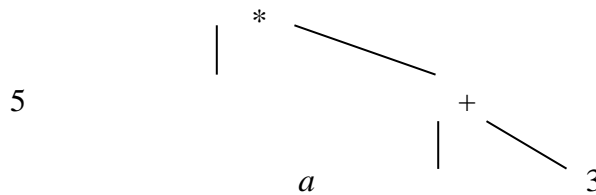


Рис. 1. Дерево-формула

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).
- преобразовать дерево-формулу  $t$ , заменяя в нем все поддеревья, соответствующие формулам  $(f_1 * (f_2 + f_3))$  и  $((f_1 + f_2) * f_3)$ , на поддеревья, соответствующие формулам  $((f_1 * f_2) + (f_1 * f_3))$  и  $((f_1 * f_3) + (f_2 * f_3))$ ;

## **2. РЕАЛИЗАЦИЯ ЗАДАЧИ**

### **Используемая структура данных для решения задачи:**

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.

### **Функции, используемые для решения задачи:**

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.

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

isSignal – 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.

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

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

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

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

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

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

PrintTree – получает дерево, делает его визуализацию.

### 3. ТЕСТИРОВАНИЕ

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

Входное выражение	Вывод программы	Корректность выполнения
(f + g)	Tree built: ( a + b ) Multiplication brackets are opened ( a + b ) Go To Prefix Form + a b	да
(a ++ b)	error: extra sign in formula Tree built: ( a + b ) Multiplication brackets are opened ( a + b ) Go To Prefix Form + a b	да

$((f + g) * a)$	Tree built: $((f + g) * a)$ Multiplication brackets are opened $((f * a) + (g * a))$ Go To Prefix Form $+ * f a * g a$	да
-----------------	---	----

#### 4 . ВЫВОД

В ходе решения поставленной задачи были освоены структуры бинарного дерева, базированного на классе. А также был повторен материал рекурсивного обхода, в данном случае дерева и запись дерева в префиксной форме.

## ПРИЛОЖЕНИЕ А

### ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл main.cpp:

```
#include <iostream>
#include <fstream>
#include <fstream>
#include <cstdlib>
#include <cstring>
#include "Btree.h"
#include "func.h"
using namespace std ;
using namespace binTree_modul;
int main () {
    string exp;
    binTree tr;
    int k = 0;
    cout << "Enter expression: ";
    tr = MakeTree(cin, 0);
    if (tr == NULL){
        //cout<<"Expression not wrote"<<endl;
        return 0;
    }
    cout << "Tree built:" << endl;
    exp = MakeForm(tr);
    cout << exp << endl;
    cout << "Visualization: " << endl;
    PrintTree(tr, 0);
    cout<<endl;
    cout << "Multiplication brackets are opened" << endl;
    Invariant(tr);
    exp = MakeForm(tr);
    cout << exp << endl;
    cout << "Go To Prefix Form" << endl;
    exp = ToPrefix(tr);
    cout << exp << endl;
    cout << "Visualization: " << endl;
    PrintTree(tr, 0);
    destroy (tr);
}
```

```

    cout << endl;
    return 0;
}

```

Файл func.cpp:

```

#include <iostream>
#include <fstream>
#include <fstream>
#include <cstdlib>
#include <cstring>
#include "Btree.h"
using namespace std;
using namespace binTree_modul;

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

bool isSignal( const char c ){
    return c == '-' || c == '+' || c == '*';
}

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

void DelSpace(istream &in){

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

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

```

```

binTree MakeTree(istream &in, int k){

```

```

    base c, sign;
    binTree left, right;
    DelSpace(in);
    c = in.get();
    if (c=='\n'){
        cout<<"Expression not wrote"<<endl;
        return NULL;
    }
    if (isSymbol(c) && k==0){ // если формула является символом
        base h;
        k++;
        DelSpace(in);
        h = in.get();
        if (h=='\n')
            return MakeLeaf(c);
        else{
            cout<<"Exp error"<<endl;
            return NULL;
        }
    }
}

```

```

    }
    if (isSymbol(c)){
        k=1;
        return MakeLeaf(c);
    }
    if (c == '('){ // если формула имеет вид (<симв><знак><симв>)
        DelSpace(in);
        k++;
        left = MakeTree(in, k++); // первая формула
        DelSpace(in);
        c = in.get();
        if (isSignal(c))
            sign = c; // знак
        else { cerr << "error: sign expected" << endl; return NULL; }
        // если после первой формулы нет знака
        DelSpace(in);
        right = MakeTree(in, k++); // вторая формула
        DelSpace(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 (isSignal(c)) cerr << "error: extra sign in formula" <<
endl;

        else cerr << "error: external symbol if formula" << endl;
        return MakeTree(in, k++);
    }
}

void Remake(binTree b){

    //(f1 * (f2 + f3))
    binTree f1, f2, f3, f4, k;
    if (b->rt->info=='+'){
        f1 = b->rt;

```

```

        f2 = b->lt;
        f3 = b->rt->rt;
        f4 = b->rt->lt;
        b->lt = ConsBT('*', f2, f4);

        b->info = '+';
        b->rt->info = '*';
        b->rt->lt=b->lt->lt;
        return;
    }
    // ((f2 + f3)*f1)
    if (b->lt->info=='+'){
        f1 = b->lt;
        f2 = b->rt;
        f3 = b->lt->lt;
        f4 = b->lt->rt;
        b->rt = ConsBT('*', f3, f2);
        b->lt->info = '*';
        b->info = '+';
        b->lt->lt = f4;
        b->lt->rt = b->rt->rt;

        return;
    }
}

void Invariant(binTree b){

    if (b == NULL)
        return;
    Invariant(b->lt);
    Invariant(b->rt);
    if (b == NULL || b->lt == NULL || b->rt == NULL)
        return;
    if ((b->lt->info != '+' && b->rt->info != '+') || b->info != '*')
        return;
    Remake(b);
    return;
}

```

```

string ToPrefix(binTree b){

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

```

```

void PrintTree(binTree f, int l){

    if(f == nullptr){
        for(int i = 0; i < l; i++)
            cout << "\t";
        cout << '#' << endl;
        return;
    }
    PrintTree(f->rt, l+1);
    for(int i = 0; i < l; i++)
        cout << "\t";
    cout << f->info << endl;
    PrintTree(f->lt, l+1);
}

```

Файл func.h:

```

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

```

```

using namespace std;
using namespace binTree_modul;
    bool isSymbol( const char c );
    bool isSignal( const char c );
    void DelSpace(istream &in);
    string MakeForm(binTree b);
    binTree MakeTree(istream &in, int k);
    void Remake(binTree b);
    void Invariant(binTree b);
    string ToPrefix(binTree b);
    void PrintTree(binTree f, int l);

```

Файл bt\_func.cpp:

```

#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)
    {
        if (b == NULL) { cerr << "Error: RootBT(null) \n"; exit(1); }
        else return b->info;
    }
//-----
    binTree Left (binTree b)
    {
        if (b == NULL) { cerr << "Error: Left(null) \n"; exit(1); }
        else return b ->lt;
    }
//-----
    binTree Right (binTree b)

```

```

        {    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;
        }
    }

}

```

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);
}

```

Файл Makefile:

```

all: main.o bt_func.o func.o
    g++ main.o bt_func.o func.o -o tree
work_bt.o: main.cpp Btree.h func.h
    g++ -c main.cpp
functions.o: func.cpp func.h Btree.h
    g++ -c func.cpp
bt_implementation.o: bt_func.cpp Btree.h
    g++ -c bt_func.cpp
clean:
    rm *.o bt_func

```