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

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Алгоритмы и структуры данных»

Тема: Деревья

Студент гр. 7383 _____ Зуев Д.В.

Преподаватель _____ Размочева Н.В.

Санкт-Петербург

2018

ОГЛАВЛЕНИЕ

Цель работы

Цель работы: познакомиться со структурой и реализацией бинарного дерева и леса на базе указателей на языке программирования C++.

Формулировка задачи: Вариант 6(а,б,в)-д. Для заданного бинарного дерева с произвольным типом элементов:

а) получить лес, естественно представленный этим бинарным деревом;

б) вывести изображение бинарного дерева и леса;

в) перечислить элементы леса в горизонтальном порядке (в ширину).

Реализация задачи

В данной работе используются класс BTree — бинарное дерево на базе указателей с шаблонным типом элементов T.

Переменные класса BTree:

- BTree* left — указатель на левое поддерево.
- BTree* right — указатель на правое поддерево.
- T node — корень данного бинарного дерева.

Методы класса Stack:

- BTree() — конструктор дерева. Инициирует правое и левое поддерево нулевыми указателями.
- ~Btree() — деструктор дерева. Удаляет правое и левое поддерева.
- T Root() — возвращает корень дерева.
- BTree* Left() — возвращает указатель на левое поддерево.
- BTree* Right() — возвращает указатель на правое поддерево.
- BTree* cons(T ,BTree* ,BTree*) — создает новое дерево из двух деревьев и элемента.

В работе используется шаблонная структура tree. Элементы структуры:

- T node — корень дерева.
- Forest<T>* f — лес состоящий из поддеревьев корня.

В работе используется используется шаблонный класс Forest — лес на базе указателей.

Переменные класса Forest:

- tree<T>* t — первое дерево леса.
- Forest* f — остальные деревья леса.

Методы класса Forest:

- Forest() - конструктор леса.
- ~Forest() - деструктор леса.
- Forest* tail_forest() - возвращает лес без первого дерева.
- Forest* current_forest() - возвращает лес состоящий из поддеревьев первого дерева.

- T root() - возвращает корень первого дерева.
- Forest* create_forest(tree<T>* ,Forest*) - создает лес из дерева и леса.
- tree<T>* create_tree(T , Forest<T>*) - создает дерево из элемента и леса.
- Forest* sum(Forest<T>*, Forest<T>*) - создает лес из двух лесов.

В функции main выводится приглашение выбрать способ ввода входных данных либо выйти из программы. В случае выбора файла, программа считывает текст из файла и записывает его в поток ввода. В случае ввода информации с консоли функция main считывает строку с консоли, записывает эту строку в этот же поток ввода.

Функция read_binTree получает на вход поток входных данных. Бросает ошибку если строка пустая. Считывает строку, если встретит переменную с типом данных, не совпадающим с введенным в начале, бросает ошибку. Записывает считанную переменную в корень и вызывает саму себя для создания левого и правого поддерева.

Функции print_Btree и print_Forest выводят бинарное дерево и лес соответственно на экран с поворотом на 90 градусов против часовой стрелки.

Функция print_width выводит элементы леса при обходе его в ширину.

Функция BinTree_to_Forest преобразует бинарное дерево в лес по принципу: корень левого поддерева — левый сын, корень правого поддерева — правый брат.

Функция print_Trees вызывает поочередно функции вывода деревьев или их элементов на экран.

Так же были реализованы классы ошибок. Каждый класс — под отдельную ошибку.

Тестирование

Программа была собрана в компиляторе G++ в среде разработки Qt creator в OS Linux Ubuntu 16.04 LTS.

В ходе тестирования была найдена ошибка. При вызове функции print_width происходила ошибка сегментации. Это происходило из-за того что функция завершала выполнение при условии не вызова самой себя. Исправлено удалением «else».

Корректные тестовые случаи представлены в приложении А.

Вывод

В ходе работы были получены навыки работы со бинарным деревом и лесом на базе указателей на языке C++. Были созданы классы бинарного дерева и леса и написана программа, выводящая бинарное дерево, лес и элементы леса при обходе в ширину.

ПРИЛОЖЕНИЕ А.

ТЕСТОВЫЕ СЛУЧАИ

Таблица 1 — Корректные случаи

Входные данные	Выходные данные
2	BinTree: 324.567 345 43.5 2 1.2 4.356 3 0 2.3 34.999 Forest: 324.567 345 43.5 2 1.2 4.356 3 0 2.3 34.999 The BFS: | 1.2 || 2 || 345 || 324.567 || 3 || 4.356 || 43.5 || 2.3 || 0 || 34.999 |
1 int (1(g)(h))	An item with invalid type was encountered.
1 char (1(g)(h))	BinTree: h 1

	g Forest: h 1 g The BFS: | 1 || h || g |
1 Введите тип данных: double Введите формулу: (1(2.3(3)(3.78))(45#(98.6)))	BinTree: 98.6 45 1 3.78 2.3 3 Forest: 98.6 45 1 3.78 2.3 3 The BFS: | 1 || 45 || 98.6 || 2.3 || 3.78 || 3 |
1 int ((2)(3))	Missing node.
1 iny (1(2)(3))	Type is not correct.
1 int	Input stream is empty.
1 x+z)	The input string: x + z) Missing opening bracket. Formula is wrong.
w	Wrong character. The character "w" is not int.
13	You entered wrong number.

ПРИЛОЖЕНИЕ Б.

ИСХОДНЫЙ КОД ПРОГРАММЫ

main.cpp:

```
#include <iostream>
#include <sstream>
#include <fstream>
#include <cstring>
#include <exception>
#include <cstdlib>

using namespace std;

class error5: public exception
{
public:
    explicit error5(const char* a)
        {this->a=a; }

    virtual const char* what() const throw()
    { return "Wrong character.\n"; }
    void printErr()
    {
        cout<<"The character \""<<a<<"\" is not int.\n";
    }
private:
    const char* a;
};

/*-------------------------БИНАРНОЕ
ДЕРЕВО---------------------------------------*/
template <typename T>
class BTree
{
private:
    BTree* left;
    BTree* right;
    T node;
public:
    BTree(){left = NULL; right = NULL;}
    ~BTree(){delete left; delete right;}
    T Root();
    BTree* Left();
    BTree* Right();
    BTree* cons(T ,BTree* ,BTree* );
};
template<typename T>
T BTree<T>::Root()
{
    return node;
```

```
}
template<typename T>
BTree<T>* BTree<T>::Left(){
    return left;
}
template<typename T>
BTree<T>* BTree<T>::Right()
{
    return right;
}
template<typename T>
BTree<T>* BTree<T>::cons(T root,BTree* BTl,BTree* BTr)
{
    BTree<T>* BT;
    BT = new BTree<T>;
    BT->node = root;
    BT->left = BTl;
    BT->right = BTr;
    return BT;
}
template <typename T>
BTree<T>* read_binTree(stringstream &xstream)
{
    T t;
    BTree<T>* BT;
    BT = new BTree<T>;
    char x;
    if(!(xstream>>x))
        throw logic_error("Input stream is empty.\n");
    if(x == '#')
        return NULL;
    if(x == '(')
    {
        if(!(xstream>>t))
            throw logic_error("Missing node.\n");
        if(xstream.peek() != '(' && xstream.peek() != ')' &&
xstream.peek() != '#')
            throw invalid_argument("An item with invalid type
was encountered.\n");
        BTree<T>* left;
        BTree<T>* right;
        if(xstream.peek()==')')
        {
            left = NULL;
            right = NULL;
        }
        else
        {
            left = read_binTree<T>(xstream);
            right = read_binTree<T>(xstream);
        }
        xstream>>x;
```

```
        if (!(x = ')')) throw invalid_argument("Missing closing
backet!\n");
        BT = BT->cons(t, left, right);
    }
    else throw invalid_argument("Missing opening bracket!\n");
    return BT;
}
template<typename T>
int print_BTree(BTree<T>* BT, int t)
{
    if(BT != NULL)
    {
        t++;
        print_BTree(BT->Right(),t);
        for(int j = 0; j<t-1;j++)
            cout<<'\t';
        cout<<BT->Root()<<endl;
        print_BTree(BT->Left(),t);
    }
    else
        cout<<endl;
    return 0;
}
/*-------------------------------------
ЛЕС-----------------------------------------*/
template<typename T>
class Forest;

template<typename T>
struct tree{
    T node;
    Forest<T>* f;
};

template<typename T>
class Forest
{
private:
    tree<T>* t;
    Forest* f;
public:
    Forest()
    {
        t = NULL;
        f = NULL;
    }
    ~Forest(){delete t; delete f;}
    Forest* tail_forest();
    Forest* current_forest();
    T root();
    Forest* create_forest(tree<T>* ,Forest* );
    tree<T>* create_tree(T , Forest<T>* );
```

```
    Forest* sum(Forest<T>*, Forest<T>* );

};
template<typename T>
Forest<T>* Forest<T>::tail_forest()
{
    return f;
}
template<typename T>
Forest<T>* Forest<T>::current_forest()
{
    return t->f;
}
template<typename T>
T Forest<T>::root()
{
    return t->node;
}
template<typename T>
Forest<T>* Forest<T>::create_forest(tree<T>* t,Forest* f)
{
    Forest* fn;
    fn = new Forest;
    fn->t = t;
    fn->f = f;
    return fn;
}
template<typename T>
tree<T>* Forest<T>::create_tree(T node, Forest<T>* f)
{
    tree<T>* t;
    t = new tree<T>;
    t->node = node;
    t->f = f;
    return t;
}
template<typename T>
Forest<T>* Forest<T>::sum(Forest<T>* f1, Forest<T>* f2)
{
    Forest<T>* f;
    f = new Forest<T>;
    if(f1 != NULL)
        f = create_forest(f1->t, sum(f1->f,f2));
    else
        f = f2;
    return f;
}
template<typename T>
int print_Forest(Forest<T>* F, int t)
{
    if(F != NULL)
    {
```

```
        if(F->tail_forest() != NULL)
            print_Forest(F->tail_forest(), t);
        for(int j = 0; j<t;j++)
            cout<<'\t';
        cout<<F->root()<<endl;
        t++;
            print_Forest(F->current_forest(), t);
    }
    return 0;
}
template<typename T>
int print_width(Forest<T>* f1)
{
    Forest<T>* f2;
    f2 = NULL;
    while(f1 != NULL)
    {
        cout<<"| "<<f1->root()<<" |";
        f2 = f2->sum(f2,f1->current_forest());
        f1 = f1->tail_forest();
    }
    if(f2!=NULL)
        print_width(f2);
    return 0;
}
/*-------------------------ОСТАЛЬНЫЕ-
ФУНКЦИИ-------------------------------------*/
template<typename T>
Forest<T>* BinTree_to_Forest(BTree<T>* BT)
{
    tree<T>* TR;
    TR = new tree<T>;
    Forest<T>* F;
    F = new Forest<T>;
    if(BT->Left()!=NULL)
        TR = F->create_tree(BT->Root(),BinTree_to_Forest(BT-
>Left()));
    else
        TR = F->create_tree(BT->Root(), NULL);
    if(BT->Right() != NULL)
        F = F->create_forest(TR, BinTree_to_Forest(BT-
>Right()));
    else
        F = F->create_forest(TR, NULL);
    return F;
}

template<typename T>
void print_Trees(BTree<T>* BT, Forest<T>* F)
{
    cout<<"BinTree:\n";
    print_BTree(BT, 0);
```

```
    cout<<"Forest:\n";
    print_Forest(F, 0);
    cout<<"The BFS:\n";
    print_width(F);
    cout<<endl;
}

int main()
{
    stringstream xstream;

    short int tmp = 0;
    while(tmp != 3)
    {
        string type;
        string str0;
        string tmp1;
        try{
            xstream.str("");
            xstream.clear();
            cout<<"Введите 1, если желаете вводить выражение с
клавиатуры.\n"
              "Введите 2, если желаете брать выражение из файла
test.txt.\n"
              "Введите, 3 если хотите закончить работу."<<endl;
            getline(cin, tmp1);
            if(!atoi(tmp1.c_str() ))
                throw error5(tmp1.c_str());
            else tmp = atoi(tmp1.c_str());
            switch(tmp){
                case 1:
                {
                    cout<<"Введите тип данных:"<<endl;
                    getline(cin,type);
                    cout << "Введите формулу: \n";
                    getline(cin, str0);
                    xstream << str0;
                    break;
                }
                case 2:
                {
                    ifstream outfile;
                    outfile.open("test.txt");
                    if (!outfile)
                        throw runtime_error("File is not
open.\n");
                    getline(outfile, type);
                    getline(outfile, str0);
                    outfile.close();
                    xstream << str0;
                    break;
                }
```

```
                case 3:
                    continue;
                default:
                    throw invalid_argument("You entered wrong
number.\n");;
            }
            if(type == "char")
            {
                BTree<char>* BT;
                BT = new BTree<char>;
                BT = read_binTree<char>(xstream);
                Forest<char>* F;
                F = BinTree_to_Forest(BT);
                print_Trees(BT,F);
                continue;
            }
            if(type == "int")
            {
                BTree<int>* BT;
                BT = new BTree<int>;
                BT = read_binTree<int>(xstream);
                Forest<int>* F;
                F = BinTree_to_Forest(BT);
                print_Trees(BT,F);
                continue;
            }
            if(type == "float")
            {
                BTree<float>* BT;
                BT = new BTree<float>;
                BT = read_binTree<float>(xstream);
                Forest<float>* F;
                F = BinTree_to_Forest(BT);
                print_Trees(BT,F);
                continue;
            }
            if(type == "double")
            {
                BTree<double>* BT;
                BT = new BTree<double>;
                BT = read_binTree<double>(xstream);
                Forest<double>* F;
                F = BinTree_to_Forest(BT);
                print_Trees(BT,F);
                continue;
            }
            throw invalid_argument("Type is not correct.\n");
        }
        catch(error5& e)
        {
            cout<<e.what();
            e.printErr();
```

```
            continue;
        }
        catch(exception& e)
        {
            cout<<e.what();
            continue;
        }
    }
    return 0;
}
```