

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

**ОТЧЕТ**  
**по лабораторной работе №5**  
**по дисциплине «Алгоритмы и структуры данных»**  
**Тема: Бинарные деревья поиска**

Студент гр. 7383

\_\_\_\_\_

Корякин М.П.

Преподаватель

\_\_\_\_\_

Размочаева Н.В.

Санкт-Петербург

2018

## Содержание

Цель работы.....	3
Реализация задачи.....	3
Тестирование .....	4
Вывод .....	4
Приложение А. Код программы .....	5
Приложение Б. Тестовые случаи .....	11

## Цель работы

Познакомиться с рандомизированными деревьями поиска и научиться реализовывать их на языке программирования C++.

По заданному файлу F (типа file of Elem), все элементы которого различны, построить дерево поиска. Записать в файл элементы построенного БДП в порядке их возрастания; вывести построенное БДП на экран.

## Реализация задачи

- Оба поддерева — левое и правое — являются двоичными деревьями поиска.
- У всех узлов левого поддерева произвольного узла X значения ключей данных меньше, нежели значение ключа данных самого узла X.
- У всех узлов правого поддерева произвольного узла X значения ключей данных больше либо равны, нежели значение ключа данных самого узла X.

В данной работе было написано несколько функций и класс для работы с деревом поиска.

class BST— структура, представляющая узел БДП, содержит в себе поля int key для хранения ключа, BST\* left, right для хранения указателей на правое и левое поддерево.

int Root — функция возвращающая значение ключа данного узла, принимающая на вход узел.

BST\* destroy — функция, которая высвобождает память, выделенную под дерево, и возвращает нулевой указатель.

BST\* Left — функция возвращающая левое поддерево узла.

BST\* Right — функция возвращающая правое поддерево узла.

BST\* BuildBST — функция, строящее дерево, и возвращающая указатель на созданный узел.

BST\* find — функция, которая проверяет есть ли в дереве узел с таким же ключем, который был подан на вход функции.

void printtree — функция, которая выводит на экран визуализацию дерева поиска.

`int comp` – функция-компаратор, для сортировки ключей узлов в порядке возрастания, созданного дерева.

`int main` – головная функция, в которой осуществляется ввод списка ключей дерева или же выбирается вариант с считывания списка из файла. После этого считанный список разбивается на строки и преобразовывается в тип `int`, с помощью функции `atoi()`. По полученным значениям строятся деревья и выводится на экран, пока пользователь не захочет завершить программу.

### **Тестирование**

Программа собрана в операционной системе Ubuntu Mint 18.01 с использованием компилятора `g++`. В других ОС и компиляторах тестирование не проводилось. Результаты тестирования показали, что поставленная цель выполнена. Результаты тестирования представлены в Приложении Б.

### **Вывод**

В ходе выполнения лабораторной работы были изучены основные понятия о деревьях поиска, была реализовано бинарное дерево поиска на языке программирования `C++`. Также была написана дополнительная функция–компаратор для сортировки ключей, для создание отсортированного дерева по возрастанию.

## ПРИЛОЖЕНИЕ А.

### ИСХОДНЫЙ КОД ПРОГРАММЫ

```
#include <iostream>
#include <vector>
#include <cstring>
#include <fstream>
#include <sstream>
using namespace std;
//template <class T>
class BST{
private:
    int key;
    BST* right;
    BST* left;
public:
    BST(int k){
        key = k;
        left = right = nullptr;
    }
    int Root(BST* b){
        if (b == NULL)
            exit(1);
        else
            return b->key;
    }
    BST* destroy(BST* tree){
        if (left)
            delete tree->left;
        if (right)
            delete tree->right;
        delete tree;
        return tree = NULL;
    }
    BST* Left(BST* b){
        if (b == NULL)
            exit(1);
        else
            return b->left;
    }
    BST* Right(BST* b){
        if (b == NULL)
            exit(1);
        else
            return b->right;
    }
    BST* BuildBST(BST* tree, int keyint){
```

```

        if (!tree)
            return new BST(keyint);
        if( tree->key > keyint )
            tree->left = BuildBST(Left(tree), keyint);
        else
            tree->right = BuildBST(Right(tree), keyint);
        return tree;
    }

    BST* find( BST* tree, int key){
        if(!tree)
            return NULL;
        if(key == tree->key)
            return tree;
        if(key < tree->key)
            return find(tree->left, key);
        else
            return find(tree->right, key);
    }
};

void printtree(BST* treenode, int l){
    if(treenode==NULL){
        for(int i = 0;i<l;++i)
            cout<<"\t";
        cout<<'# '<<endl;
        return;
    }
    printtree(treenode->Right(treenode), l+1);
    for(int i = 0; i < l; i++)
        cout << "\t";
    cout << treenode->Root(treenode)<< endl;
    printtree(treenode->Left(treenode),l+1);
}

int comp (const int *i, const int *j)
{
    return *i - *j;
}

int main(){
    BST *b = NULL;
    BST *p = NULL;
    string exp;
    stringbuf temp;
    int k;
    int arrsize;
    while(1){
        int out = 0;
        cout<<"\nВведите 1 - если хотите считать/ь из файла, 2 - если
хотите выполнить ввод с консоли, 0 - если хотите завершить
программу\n"<<endl;
        cin>>k;
        switch(k){

```

```

        case 1:{
            ifstream infile("1.txt");
            if(!infile){
                cout<<"There is no file"<<endl;
                continue;
            }
            getline(infile, exp);}
        break;
    case 2:{
        cout << "Введите список:" << endl;
        cin.ignore();
        getline(cin, exp);}
        break;
    case 0:{
        cout<<"Программа завершилась"<<endl;
        out = 1;}
        break;
    default:
        cout<<"Вы ввели какую-то херню, попробуйте еще
раз"<<endl;
        break;
    }
    if (out == 1)
        break;
    int keyint;
    int count=0;
    int* arr = new int;
    char* OneKey;
    char* StrKeys = new char[exp.size()+1];
    strcpy(StrKeys, exp.c_str());
    OneKey = strtok(StrKeys, " ");
    while (OneKey != NULL){

        keyint = atoi(OneKey);

        arr[count] = keyint;
        count++;
        cout<<arr[count-1]<<endl;
        if(b->find(b, keyint)){
            OneKey=strtok(NULL, " ");
            continue;
        }
        b = b->BuildBST(b, keyint);

        OneKey = strtok (NULL, " ");
    }
    printtree(b, 0);
    qsort(arr, count, sizeof (int), (int(*) (const void *, const void
*)) comp);
    int i;
    for (i=0; i<count; i++){

```

```

        p = p->BuildBST(p, arr[i]);
    }
    cout<<"Дерево отсортировано по возрастанию\n"<<endl;
    printtree(p, 0);
    p = p->destroy(p);
    delete arr;
    exp.clear();
    b = b->destroy(b);
    delete [] StrKeys;
    delete OneKey;

    }
    return 0;
}

```



## ПРИЛОЖЕНИЕ Б.

### Тестовые случаи.

Результаты тестов представлены на рис 1-3.

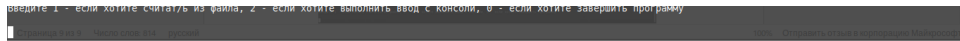


Рисунок 1

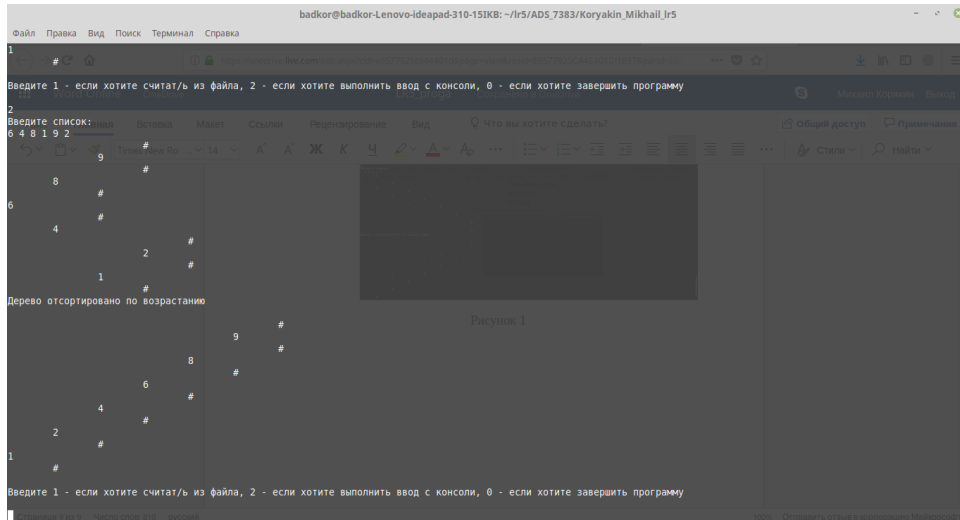


Рисунок 2

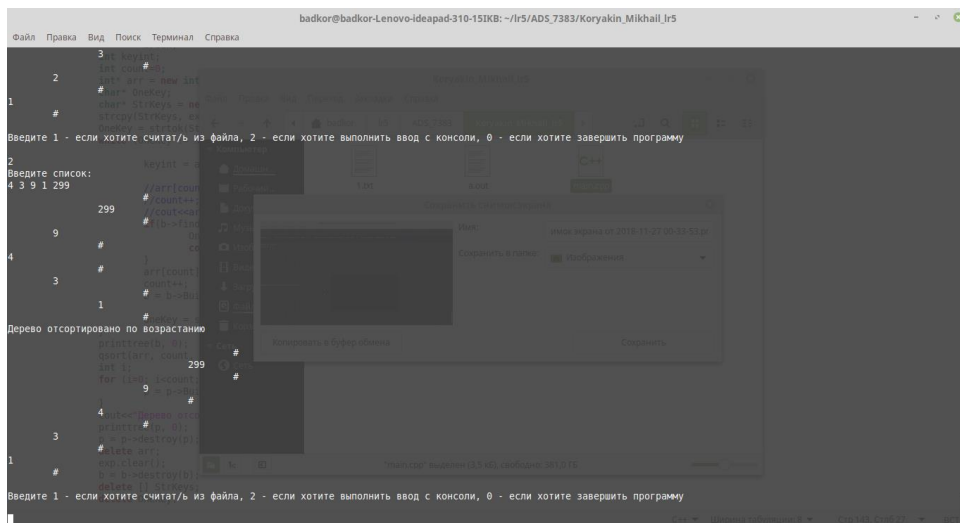


Рисунок 3