Министерство образования и науки РФ
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
Санкт-Петербургский государственный
электротехнический университет «ЛЭТИ» имени В.И. Ульянова (Ленина)

Факультет компьютерных технологий и информатики
Кафедра вычислительной техники

Зачётная работа №3
по дисциплине «Алгоритмы и структуры данных»
«Деревья»

Вариант№21

Выполнил: студент группы 5005
Зинатуллин Азат Салаватович

Проверил: старший преподаватель
Колинько Павел Георгиевич

Цель работы

Получить практические навыки в работе с классами. Научится использовать алгоритмы обхода деревьев.

1. Задание

Нужно вычислить высоту правого поддерева для корня.
Вид дерева: Двоичное.
Разметка: Прямая.
Способ обхода: Внутренний (симметричный).

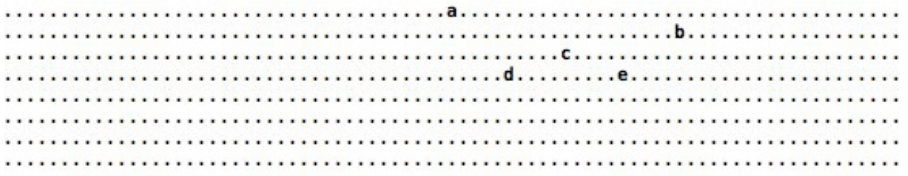
2. Обоснование выбора способа представления дерева в памяти.

Для представления дерева в памяти можно предложить естественный способ — разветвляющийся список. Узлы дерева — объекты, связи между которыми осуществляются через указатели. Это наиболее удобный способ для работы с деревьями, так как мы легко можем создать и удалить узел, обойти все дерево, разными способами. Используя массивы, мы конечно можем представить дерево, но такой гибкости как со списком не добьемся.

3. Результаты прогона программы с генерацией случайного.

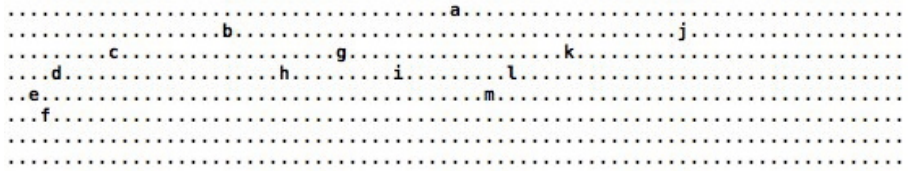

```
Обход в глубину: b_f_e_d_g_c_a_ Пройдено узлов = 7
Высота правого поддерева для корня = 0
=== Конец ===
```

Тест 1.




```
Обход в глубину: a_d_c_e_b_ Пройдено узлов = 5
Высота правого поддерева для корня = 3
=== Конец ===
```

Тест 2.




```
Обход в глубину: e_f_d_c_b_h_g_i_a_m_l_k_j_ Пройдено узлов = 13
Высота правого поддерева для корня = 4
=== Конец ===
```

Тест 3.

4. Оценка временной сложности при работе с деревьями.

Функция	Создание дерева	Обход дерева	Подсчет высоты
Временная сложность	O(n)	O(n)	O(n)

5. Выводы

В данной программе для внутреннего обхода дерева используем рекурсию. Количество шагов рекурсивного обхода зависит от количества вершин дерева, а каждая вершина обрабатывается за постоянное время, следовательно, все реализованные в программе обходы дерева имеют линейную сложность.

6. Список использованных источников

1. Алгоритмы и структуры данных: методические указания к лабораторным работам, практическим занятиям и курсовому проектированию. Ч. 1. Вып. 1701 (для заочников) / сост.: П. Г. Колинько. — СПб.: Изд-во СПбГЭТУ «ЛЭТИ», 2017. — 64 с.

2. Курс видео уроков по C++. https://www.youtube.com/playlist?list=PLbmlzoDQrXVFC13GjpPrJxl6mzTiX65gs

3. Статья: Бинарные деревья поиска и рекурсия. https://habrahabr.ru/post/267855/

4. Использовал, как справочник по некоторым функциям. http://cppstudio.com/cat/274/

7. Приложение

main.cpp - Файл создает объект класса Tree, который в свою очередь выполняет все функции по созданию, разметке, обходу деревьев. А так же расчет высоты правого поддерева для корня.
class_derevo.h - Подключаемый заголовочный файл. Содержит описания классов.

main.cpp

```
// Задание выполнил студент группы 5005 Зинатуллин Азат.
// Вариант №21. Универсум десятичные числа.
// Вид дерева: двоичное. Разметка: прямая. Способ обхода: внутренний.
// Нужно вычислить: высоту правого поддерева для корня.

#include <iostream>
#include <ctime>
#include "class_derevo.h"


using namespace std;


int main(int argc, const char * argv[])
{
        int n = 0;
        Tree Tr('a','z',8);
        srand(time(nullptr));
        setlocale(LC_ALL, "Russian");

        Tr.MakeTree();

        if(Tr.exist())
        {
            Tr.OutTree();
            cout << "\n" << "Обход в глубину: ";
            n= Tr.SFS();
            cout << " Пройдено узлов = " << n;
            Tr.outDepth();
        }
        else cout << "Дерево пусто!";
        cout << "\n" << "=== Конец ===";

    return 0;
}
```

class_derevo.h

```
#ifndef Derevja_class_derevo_h
#define Derevja_class_derevo_h

#include <iostream>
#include <cstdlib>

using namespace std;

//Класс «узел дерева»
class Node
{


public:

    char d;     // тег узла
    Node * lft; // левый сын
    Node * rgt; // правый сын


    Node() : lft(nullptr), rgt(nullptr) { } // конструктор узла
    ~Node(){ if(lft) delete lft;            // деструктор (уничтожает поддерево)
        if (rgt) delete rgt; }

    friend class Tree;          // дружественный класс «дерево»
};


// Класс «дерево в целом»
class Tree
{
    Node * root; // указатель на корень дерева
    int count;
    char num, maxnum;       //счётчик тегов и максимальный тег
    int maxrow, offset;     //максимальная глубина, смещение корня
    char ** SCREEN;         // память для выдачи на экран
    void clrscr();          // очистка рабочей памяти
    Node* MakeNode(int depth);      // создание поддерева
    void OutNodes(Node * v, int r, int c);         // выдача поддерева

    Tree (const Tree &);        // фиктивный конструктор копии
    Tree (Tree &&);             //копия с переносом (C++11)
    Tree operator = (const Tree &) const;   // фиктивное присваивание
    Tree operator = (Tree &&) const;        //присваивание с переносом (C++11)
    int SFS1(Node *v);   // Внутренний обход дерева

public:

    Tree(char num, char maxnum, int maxrow);
    ~Tree();
    void MakeTree() { root = MakeNode(0); } // ввод – генерация дерева
    bool exist() { return root != nullptr; } // проверка «дерево не пусто»
    void OutTree();     // выдача на экран

    int SFS(){ return SFS1(root); }  // Интерфейсная функция внутреннего обхода

    int getMaxDepth(Node* v, int depth); // Рекурсивная функция расчета высоты дерева

    void outDepth();            // Функция выдает высоту правого поддерева для корня

};


// Конструктор дерева инициализирует параметры разметки и создаёт рабочую память матрицу символов,
необходимую для выдачи изображения дерева на экран.
Tree::Tree(char nm, char mnm, int mxr): count(0),
num(nm), maxnum(mnm), maxrow(mxr), offset(40), root(nullptr), SCREEN(new char * [maxrow])
{for(int i = 0; i < maxrow; i++)
    SCREEN[i] = new char[80];}


// Деструктор дерева уничтожает матрицу символов и запускает деструктор узла для корня.
Tree::~Tree()
{ for(int i = 0; i < maxrow; i++) delete []SCREEN[i];
    delete []SCREEN; delete root; }

Node * Tree::MakeNode(int depth)
{
    Node * v = nullptr;
    int Y = (depth < rand()%6+1) && (num <= 'z');

    // Вариант: cout « "Node (" « num « ',' « depth « ")1/0: "; cin » Y;

    if (Y)              // создание узла, если Y = 1
    {
```

```
        v = new Node;
        v->d = num++;        // разметка в прямом порядке (= «в глубину»)
        v->lft = MakeNode(depth+1);
        // v->d = num++;           //вариант – во внутреннем
        v->rgt = MakeNode(depth+1);
        // v->d = num++;        // вариант – в обратном
    }
    return v;
}

void Tree::OutTree()
{
    clrscr();
    OutNodes(root, 1, offset);
    for (int i = 0; i < maxrow; i++)
    {
        SCREEN[i][79] = 0;
        cout << "\n" << SCREEN[i];
    }
    cout << "\n";
}

void Tree::clrscr()
{
    for(int i = 0; i < maxrow; i++)
        memset(SCREEN[i], '.', 80);
}


void Tree::OutNodes(Node * v, int r, int c)
{
    if (r&&c&&(c<80)) SCREEN[r-1][c-1]=v->d; // вывод метки
    if (r<maxrow)
    {
        if (v->lft) OutNodes(v->lft, r+1, c-(offset>>r)); //левый сын
        // if (v->mdl) OutNode(v->mdl, r + 1, c); – средний сын (если нужно)
        if (v->rgt) OutNodes(v->rgt, r+1, c+(offset>>r)); //правый сын
    }
}


int Tree::SFS1(Node *v)
{
    if(v == NULL)
    { return 0;}
    SFS1(v->lft);
    cout << v->d << '_';count++;
    SFS1(v->rgt);

    return count;
}

void Tree::outDepth()
{
    int depth = 0;
    depth = getMaxDepth(root->rgt, depth);
    cout << "\nВысота правого поддерева для корня = " << depth;
}

int Tree::getMaxDepth(Node* v, int depth)
{
    if (!v)
        return depth;
    else
        return max(getMaxDepth(v->lft, depth + 1), getMaxDepth(v->rgt, depth + 1));
}

#endif
```