**ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»**

**(СПбГУТ)**

**Кафедра безопасности информационных систем**

**ОТЧЁТ**

по практической работе № 8 на тему:   
**«Нелинейные динамические структуры данных. Бинарные деревья»**

по дисциплине «Алгоритмы и структуры данных»

Выполнил: студент группы ИСТ-312, Кандиков М.В.

«20» Ноября 2024 г. \_\_\_\_\_\_\_\_\_\_\_/М.В. Кандиков /

Принял: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«26» Ноября 2024 г. \_\_\_\_\_\_\_\_\_\_\_/ И.А. Моисеев /

**Содержание**

**1. Титульный лист 1**

**2. Содержание 2**

**3. Основная часть 3**

3.1. Цель работы 3

3.2. Теоретическая часть 3

3.3. Практическая часть 4

3.5. Выводы 8

**4. Приложение 9**

**Основная часть**

**Цель работы:**

Целью данной практической работы является закрепление теоретических знаний о бинарных деревьях поиска и идеально сбалансированных бинарных деревьях, их реализации и применении. Работа направлена на развитие навыков программирования структур данных, основанных на деревьях, их обработки, визуализации структуры и анализа особенностей.

**Теоретическая часть:**

В рамках изучения бинарных деревьев рассмотрены следующие концепции:

1. **Бинарное дерево поиска**  
   Бинарное дерево поиска (БДП) — это структура данных, где каждый узел содержит ключ, а для всех узлов левого поддерева ключи меньше ключа текущего узла, а для всех узлов правого поддерева — больше или равны. Такая структура позволяет эффективно выполнять операции поиска, вставки и удаления элементов.

Алгоритм построения бинарного дерева поиска заключается в последовательном добавлении узлов:

* Если дерево пустое, то добавляемый узел становится корнем.
* Если ключ нового элемента меньше ключа текущего узла, перемещаемся в левое поддерево.
* Если больше, — в правое поддерево.
* Если ключ совпадает, увеличиваем счётчик количества узлов с этим ключом.

Код добавления узла в бинарное дерево представлен функцией insertBST:

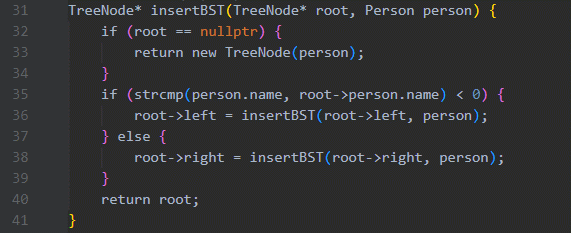


Рисунок 1. Функция insertBST.

1. **Идеально сбалансированное дерево**  
   Идеально сбалансированное дерево — это бинарное дерево, у которого разница между количеством узлов в левом и правом поддеревьях для каждой вершины не превышает одного. Алгоритм построения сбалансированного дерева предполагает равномерное распределение узлов между поддеревьями, что достигается с помощью рекурсивного деления множества узлов.
2. **Алгоритмы обхода и вывода дерева**  
   Для вывода дерева на экран использовался рекурсивный алгоритм. Узлы выводятся с учётом уровня глубины: сначала правое поддерево, затем текущий узел, и в конце — левое поддерево. Такой порядок обеспечивает визуализацию дерева, близкую к его реальной структуре.

**Практическая часть:**

1. **Считывание данных из файла:**

Программа обрабатывает входной текстовый файл lr.txt. Каждая строка файла содержит информацию о человеке: ФИО, год рождения, регион, номер телефона и группу.

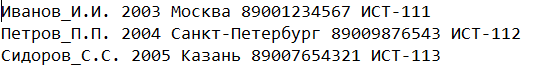


Рисунок 2. Пример строк из файла lr.txt.

Считанные данные распределяются в четыре линейных списка в зависимости от номера группы. Для хранения информации о человеке используется структура Person, которая включает необходимые поля и указатель на следующий элемент списка.

Код функции для считывания данных и распределения по спискам readFromFile выглядит следующим образом:



Рисунок 3. Функция readFromFile.

1. **Построение бинарного дерева поиска**:  
   После формирования списков программа создает бинарное дерево поиска для каждой группы. В качестве ключа для упорядочивания элементов используется поле name (ФИО). Алгоритм построения дерева реализован с использованием рекурсивной функции.

Код создания дерева представлен в функции buildAndPrintTrees.

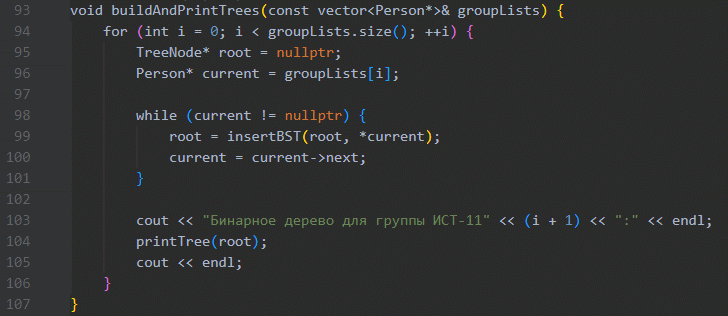


Рисунок 4. Функция buildAndPrintTrees.

1. **Результаты выполнения программы**

После запуска программы были построены бинарные деревья для каждой группы, сформированные из данных текстового файла lr.txt. Ниже представлены результаты вывода деревьев для всех четырёх групп:

**Текст файла** lr.txt**:**

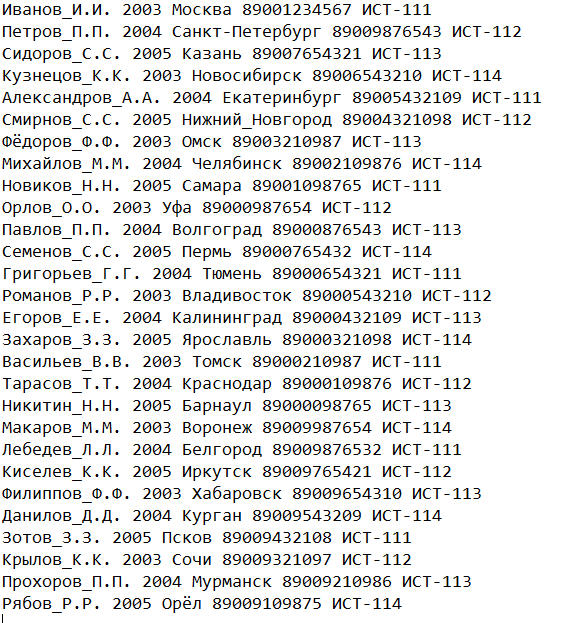


Рисунок 5. Текст файла lr.txt.

После обработки данных из файла lr.txt программа сгруппировала участников по их принадлежности к определённым группам (ИСТ-111, ИСТ-112, ИСТ-113, ИСТ-114) и создала для каждой группы бинарное дерево поиска. Эти деревья сортируют участников по фамилии в лексикографическом порядке.

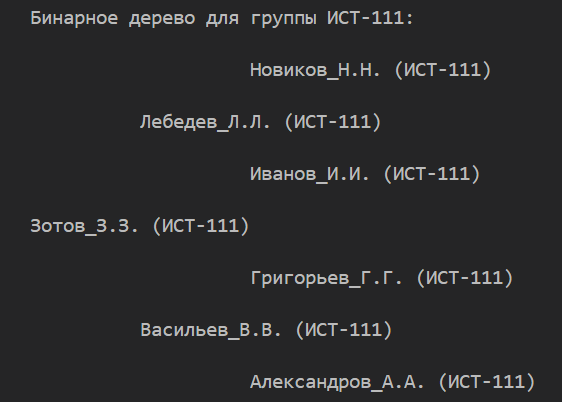


Рисунок 6. Результат выполнения программы для группы ИСТ-111.

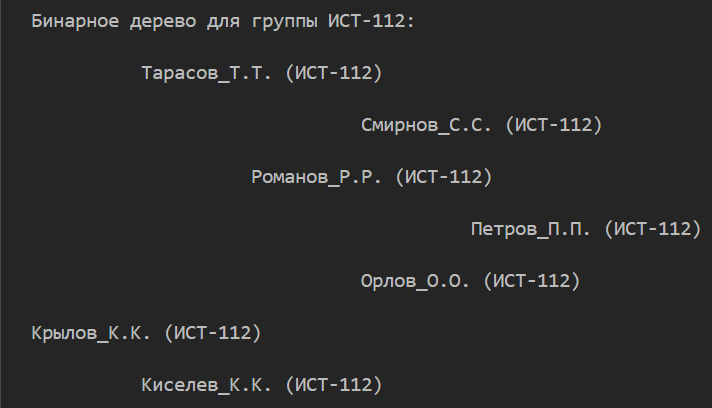


Рисунок 7. Результат выполнения программы для группы ИСТ-112.

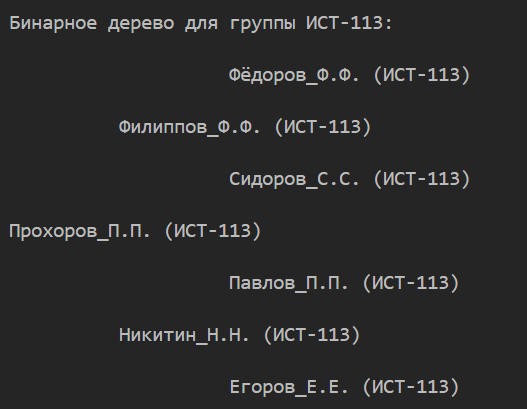


Рисунок 8. Результат выполнения программы для группы ИСТ-113.

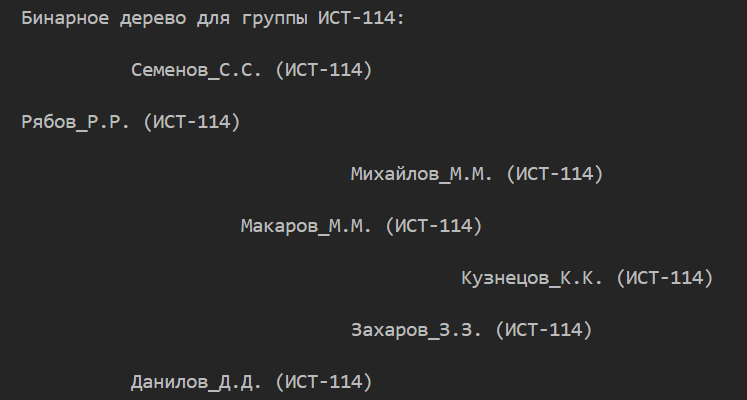


Рисунок 9. Результат выполнения программы для группы ИСТ-114.

Таким образом, программа успешно создала бинарные деревья поиска, где фамилии участников сортируются по алфавиту. Каждое дерево соответствует своей группе. Вывод наглядно демонстрирует корректность работы алгоритмов чтения данных, группировки и построения деревьев.

**Выводы:**

В ходе выполнения данной практической работы была создана программа, которая эффективно обрабатывает и структурирует данные о студентах. Используя язык C++, я реализовал считывание информации из текстового файла, распределение данных по линейным спискам в зависимости от групп и их последующую обработку через бинарные деревья поиска.

Каждая группа была представлена своим линейным списком, что упростило организацию данных. Затем эти списки преобразовывались в бинарные деревья поиска, где узлы упорядочивались по фамилиям студентов. Такой подход обеспечил удобство для визуализации и быструю сортировку данных. Вывод программы наглядно показал структуру каждого дерева с правильным алфавитным порядком.

Работа над проектом позволила закрепить знания о динамических структурах данных, таких как списки и деревья, а также углубить понимание их применения для обработки больших объемов информации. Дополнительно я освоил работу с указателями, динамической памятью и файлами в C++.

В результате была создана программа, которая соответствует требованиям задачи, демонстрирует корректную группировку и сортировку данных, а также позволяет наглядно представить результаты в удобной форме.

**Приложение**

Листинг программы:

#include <iostream>

#include <fstream>

#include <string>

#include <vector>

#include <algorithm>

#include <cstring>  // Для использования strcmp

using namespace std;

// Структура для хранения информации о человеке

struct Person {

    char name[50];      // ФИО

    int birthYear;      // Год рождения

    char region[30];    // Регион

    char phone[15];     // Номер телефона

    char group[10];     // Группа

    Person\* next;       // Указатель на следующий элемент списка

};

// Узел бинарного дерева поиска

struct TreeNode {

    Person person;

    TreeNode\* left;

    TreeNode\* right;

    TreeNode(Person p) : person(p), left(nullptr), right(nullptr) {}

};

// Функция для добавления элемента в бинарное дерево поиска по фамилии

TreeNode\* insertBST(TreeNode\* root, Person person) {

    if (root == nullptr) {

        return new TreeNode(person);

    }

    if (strcmp(person.name, root->person.name) < 0) {

        root->left = insertBST(root->left, person);

    } else {

        root->right = insertBST(root->right, person);

    }

    return root;

}

// Функция для печати бинарного дерева поиска (уровневый обход)

void printTree(TreeNode\* root, int space = 0, int height = 10) {

    if (root == nullptr) return;

    space += height;  // Увеличиваем отступ

    printTree(root->right, space);  // Печатаем правое поддерево

    cout << endl;

    for (int i = height; i < space; i++) cout << " ";  // Выводим отступы

    cout << root->person.name << " (" << root->person.group << ")" << endl;  // Печатаем узел

    printTree(root->left, space);  // Печатаем левое поддерево

}

// Функция для считывания данных из файла и создания линейных списков

void readFromFile(const string& filename, vector<Person\*>& groupLists) {

    ifstream file(filename);

    if (!file.is\_open()) {

        cout << "Не удалось открыть файл!" << endl;

        return;

    }

    string line;

    while (getline(file, line)) {

        Person\* newPerson = new Person();

        sscanf(line.c\_str(), "%s %d %s %s %s", newPerson->name, &newPerson->birthYear,

               newPerson->region, newPerson->phone, newPerson->group);

        newPerson->next = nullptr;

        // Определяем список по группе

        if (strcmp(newPerson->group, "ИСТ-111") == 0) {

            newPerson->next = groupLists[0];

            groupLists[0] = newPerson;

        } else if (strcmp(newPerson->group, "ИСТ-112") == 0) {

            newPerson->next = groupLists[1];

            groupLists[1] = newPerson;

        } else if (strcmp(newPerson->group, "ИСТ-113") == 0) {

            newPerson->next = groupLists[2];

            groupLists[2] = newPerson;

        } else if (strcmp(newPerson->group, "ИСТ-114") == 0) {

            newPerson->next = groupLists[3];

            groupLists[3] = newPerson;

        }

    }

    file.close();

}

// Функция для построения бинарного дерева для каждой группы и вывода его на экран

void buildAndPrintTrees(const vector<Person\*>& groupLists) {

    for (int i = 0; i < groupLists.size(); ++i) {

        TreeNode\* root = nullptr;

        Person\* current = groupLists[i];

        while (current != nullptr) {

            root = insertBST(root, \*current);

            current = current->next;

        }

        cout << "Бинарное дерево для группы ИСТ-11" << (i + 1) << ":" << endl;

        printTree(root);

        cout << endl;

    }

}

int main() {

    // Вектор для хранения указателей на линейные списки для каждой группы

    vector<Person\*> groupLists(4, nullptr);

    // Считывание данных из файла

    readFromFile("lr.txt", groupLists);

    // Построение и вывод бинарных деревьев поиска для каждой группы

    buildAndPrintTrees(groupLists);

    return 0;

}