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

по дисциплине «Алгоритмы и структуры данных»

Выполнил: студент группы ИСТ-114,Константинов А.Д.

«\_\_» декабря 2022 г. \_\_\_\_\_\_\_\_\_\_\_/А.Д. Константинов/

Принял: к.ф.-м.н., доцент, И.А. Моисеев

« » ноября 2022 г. \_\_\_\_\_\_\_\_\_\_\_/ И.А. Моисеев /

**Содержание:**

[Цель работы: 3](#_Toc114692552)

[Листинг кода: 3](#_Toc114692553)

[Результат работы программы: 8](#_Toc114692554)

[Вывод: 8](#_Toc114692555)

[Приложение: 22](#_Toc114692556)

Цель работы:

# Закрепить материал лекции 4

**Задание.**

Необходимо создать структуру с полями

* 1. ФИО - фамилия и инициалы, символьный массив.
  2. Год рождения - целочисленная переменная.
  3. Регион
  4. Номер телефона - символьный массив.
  5. Группа

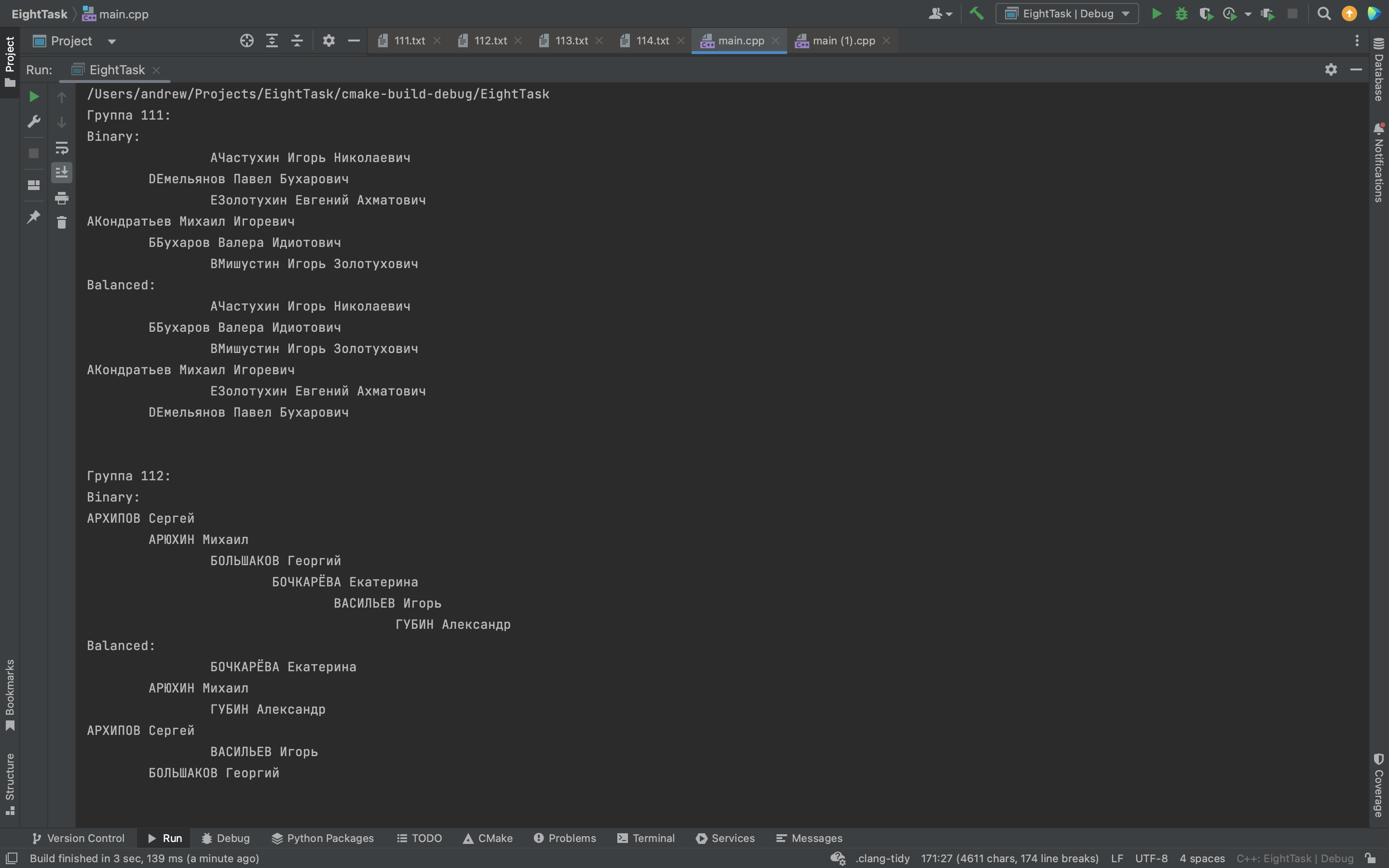
Требуется создать четыре линейных списка, состоящих из 6 таких структур. Имя каждого файла должно начинаться с номера группы.

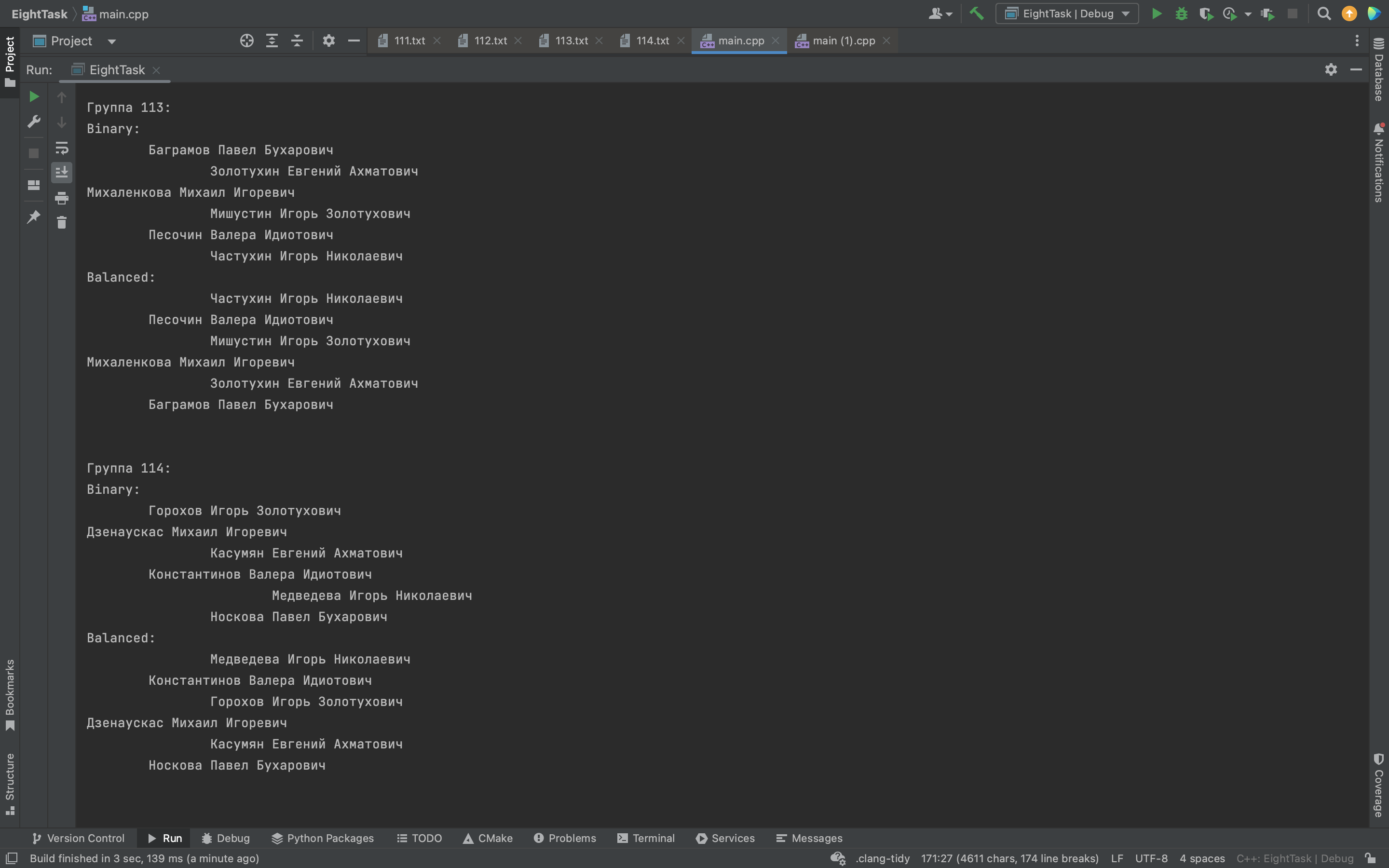
* 1. 1)  Ознакомиться с теоретическим материалом задания.
  2. 2)  Разработать программу, которая:
     1. Считывает данные из примера ранее рассматриваемого текстового файла (\*.txt) и формирует четыре линейные динамические структуры (либо связные списки, либо очереди, либо стеки), ключом формирования является номер группы (ИСТ-111, ИСТ-112, ИСТ-113, ИСТ-114). Данные считываются в том порядке, в каком они пред- ставлены в текстовом файле\*.txt.
     2. Для каждого динамического списка (для каждой группы) программа строит и вы- водит на экран бинарное дерево поиска и сбалансированное бинарное деревопо ключу Фамилия.

# Листинг кода:

#include <iostream>  
#include "fstream"  
#include "vector"  
struct Student{  
 std::string fio = "";  
 int year;  
 int region;  
 std::string phone;  
 int group;  
};  
struct Node{  
 Student student;  
 int count;  
 Node\* left = nullptr;  
 Node\* right = nullptr;  
};  
bool Compair(std::string a, std::string b){  
 int t = a.compare(b);  
 if(t >= 1) return true;  
 else return false;  
}  
int height(Node \* root){  
 if (root == nullptr) return 0;  
 return std::max(height(root->right)+1,height(root->left)+1);  
}  
int balance(Node \* root){  
 return height(root->right) - height(root->left);  
}  
void BalancedTreePush (Node\*\* root, Student newP){  
 if (\*root == nullptr){  
 Node \*newNode = new Node();  
 newNode->student = newP;  
 newNode->count = 1;  
 newNode->left = nullptr;  
 newNode->right = nullptr;  
 \*root= newNode;  
 return;  
 }  
 if (balance(\*root) == 0){  
 BalancedTreePush(&((\*root)->right), newP);  
 return;  
 }  
 else if (balance(\*root) == 1) {  
 BalancedTreePush(&((\*root)->left), newP);  
 return;  
 }  
 else{  
 BalancedTreePush(&((\*root)->right), newP);  
 return;  
 }  
}  
void BinaryTreePush(Node\*\* root, Student\* student){  
 if(\*root == nullptr){  
 Node\* newNode = new Node;  
 newNode->student = \*student;  
 newNode->right = nullptr;  
 newNode->left = nullptr;  
 newNode->count=1;  
 (\*root) = newNode;  
 }  
 else{  
 if((\*root)->student.fio == student->fio) {  
 (\*root)->count++;  
 return;  
 }  
 if(Compair((\*root)->student.fio,student->fio)) {  
// if((\*root)->student.region < student->region){  
 //right  
 BinaryTreePush(&((\*root)->right), student);  
 }  
 else {  
 //left  
 BinaryTreePush(&((\*root)->left), student);  
 }  
 }  
}  
void ReadFile(std::string fileName, Node\*\* root,Node\*\* bRoot)  
{  
 std::ifstream file(fileName);  
 std::vector<Student> list;  
 while(!file.eof())  
 {  
 Student student;  
 getline(file,student.fio);  
 std::string tmp = "";  
 getline(file,tmp);  
 student.year = std::atoi(tmp.c\_str());  
 getline(file,tmp);  
 student.region = std::atoi(tmp.c\_str());  
 getline(file,student.phone);  
 getline(file,tmp);  
 student.group = std::atoi(tmp.c\_str());  
 list.push\_back(student);  
 BinaryTreePush(root,&student);  
 BalancedTreePush(bRoot,student);  
 }  
  
}  
void Print(Node\*\* root, int l = 0){  
 if(\*root != nullptr){  
 Print(&(\*root)->right,l+1);  
 for (int i = 1; i <= l\*2; ++i) {  
 std::cout<<" ";  
 }  
 std::cout << (\*root)->student.fio << std::endl;  
 Print(&(\*root)->left,l+1);  
 }  
}  
int main()  
{  
 std::cout << "Группа 111:" << std::endl;  
 Node\* binaryNode = new Node;  
 Node\* balancedNode = new Node;  
 binaryNode = nullptr;  
 balancedNode = nullptr;  
 ReadFile("111.txt",&binaryNode,&balancedNode);  
 std::cout<< "Binary:" <<std::endl;  
 Print(&binaryNode);  
 std::cout<< "Balanced:" <<std::endl;  
 Print(&balancedNode);  
 delete(binaryNode);  
 delete(balancedNode);  
 std::cout<< std::endl;  
 std::cout<< std::endl;  
  
  
 std::cout << "Группа 112:" << std::endl;  
 binaryNode = new Node;  
 balancedNode = new Node;  
 binaryNode = nullptr;  
 balancedNode = nullptr;  
 ReadFile("112.txt",&binaryNode,&balancedNode);  
 std::cout<< "Binary:" <<std::endl;  
 Print(&binaryNode);  
 std::cout<< "Balanced:" <<std::endl;  
 Print(&balancedNode);  
 delete(binaryNode);  
 delete(balancedNode);  
 std::cout<< std::endl;  
 std::cout<< std::endl;  
  
  
 std::cout << "Группа 113:" << std::endl;  
 binaryNode = new Node;  
 balancedNode = new Node;  
 binaryNode = nullptr;  
 balancedNode = nullptr;  
 ReadFile("113.txt",&binaryNode,&balancedNode);  
 std::cout<< "Binary:" <<std::endl;  
 Print(&binaryNode);  
 std::cout<< "Balanced:" <<std::endl;  
 Print(&balancedNode);  
 delete(binaryNode);  
 delete(balancedNode);  
 std::cout<< std::endl;  
 std::cout<< std::endl;  
  
  
 std::cout << "Группа 114:" << std::endl;  
 binaryNode = new Node;  
 balancedNode = new Node;  
 binaryNode = nullptr;  
 balancedNode = nullptr;  
 ReadFile("114.txt",&binaryNode,&balancedNode);  
 std::cout<< "Binary:" <<std::endl;  
 Print(&binaryNode);  
 std::cout<< "Balanced:" <<std::endl;  
 Print(&balancedNode);  
 delete(binaryNode);  
 delete(balancedNode);  
 std::cout<< std::endl;  
 std::cout<< std::endl;  
 return 0;  
}

# Результат работы программы:





# Вывод:

В ходе выполнения практической работы была написана программа, строящая бинарное дерево поиска и идеально сбалансированное дерево