```cpp
#ifndef FILECONTROLLER_H
#define FILECONTROLLER_H

#include "binaryNode.h"
#include "coloredNode.h"
#include "tree.h"
#include "node.h"
#include "basicNotion.h"
#include <algorithm>
#include <iostream>
#include <fstream>
#include <chrono>

using namespace std;


class fileController {
private:
	tree* base;
	int size = 0;
	string binaryFilePath = "data.dat";
	string textFilePath = "data.txt";
	int lastRowNumber = 0;

public:

	fileController(tree* base) {
		this->base = base;
	}

	void execute() {
		cout << "Welcome to tree file controller\n";
		int iControlChecker = 0;

		while (iControlChecker != 7) {
			cout << "Choose program\n"
				<< "0. Generate text file \n"
				<< "1. Create binary file \n"
				<< "2. Fill tree\n"
				<< "3. Add node\n"
				<< "4. Delete node\n"
				<< "5. Find node\n"
				<< "6. Show tree\n"
				<< "7. Exit\n"
				<< "8. Test\n";
			cin >> iControlChecker;

			switch (iControlChecker)
			{
			case 0: {
				string textFilePath;
				int size = 0;
				cout << "Enter file location: ";
				cin >> textFilePath;
				cout << "Enter file size: ";
				cin >> size;
				if (size < 1) {
					cout << "Too low size, try again";
```

```
                break;
            }
            generateFile(textFilePath, size);
            break;
        }
        case 1: {
            string stTextInputFilePath;
            cout << "Enter basic text file location: \n";
            cin >> stTextInputFilePath;
            textFilePath = stTextInputFilePath;
            createBinaryFile(textFilePath);
            break;
        }
        case 2: {
            size = fillTree();
            break;
        }
        case 3: {
            string FIO;
            double GPA;
            bool isExcluded;
            cout << "Enter info about student: \n";
            cout << "FIO(1-9 symb) : ";
            cin >> FIO;
            cout << "Enter GPA: ";
            cin >> GPA;
            cout << "Enter excluded status: ";
            cin >> isExcluded;

            createBinaryFile(textFilePath);
            addNode(FIO, lastRowNumber);
            size++;
            break;
        }
        case 4: {

            if (size == 0)
            {
                cout << "Tree is empty, opeartion is not 
allowed\n";

                break;
            }

            string FIO;
            cout << "Write student's FIO: \n";
            cin >> FIO;

            deleteNode(FIO);

            createBinaryFile(textFilePath);

            size--;

            break;
        }
        case 5: {
            if (size == 0)
            {
```

```
                                        cout << "Tree is empty, opeartion is not
allowed\n";
                                        break;
                                }

                                string FIO;
                                cout << "Write student's FIO: \n";
                                cin >> FIO;
                                findNode(FIO);
                                break;
                        }
                        case 6: {
                                showTree();
                                break;
                        }
                        case 7: {
                                return;
                        }
                        case 8: {
                                /*
                                generateFile("data.txt", 10);
                                createBinaryFile("data.txt");
                                fillTree();
                                showTree();
                                addNode("cccccclm", 11);
                                showTree();
                                deleteNode("cccccccl");
                                showTree();
                                findNode("cccccccg");
                                */
                                generateFile("data.txt", 100);
                                createBinaryFile("data.txt");
                                size = fillTree();
                                findNode("cccccec"); //cccccec
                                cout << "TEST 1, size: " <<size * sizeof(ColoredNode)
<< endl;
                                generateFile("data.txt", 10000);
                                createBinaryFile("data.txt");
                                size = fillTree();
                                findNode("ccckccc"); //ccckccc
                                cout << "TEST 2, size: " << size *
sizeof(ColoredNode) << endl;
                                generateFile("data.txt", 100000);
                                createBinaryFile("data.txt");
                                size = fillTree();
                                findNode("ccclcdg");//ccclcdg
                                cout << "TEST 3, size: " << size *
sizeof(ColoredNode) << endl;
                        }
                        }
                }
        }

        void createBinaryFile(string stTextInputFilePath) {
                ifstream in(stTextInputFilePath);
                ofstream out(binaryFilePath, ios::out | ios::binary);
                notion* tmp = nullptr;
                string FIO;
                double GPA;
                bool excluded;

                while (!in.eof()) {
                        in >> FIO >> GPA >> excluded;
                        tmp = new notion(FIO, GPA, excluded);
                        out.write((char*)&(*tmp), sizeof(notion));
                }

                in.close();
                out.close();
        }

        int fillTree() {
                notion* tmp = new notion("a", 0, false);
                string tmpPrevKey = "0";
                ifstream in(binaryFilePath, ios::binary | ios::in);
                int count = 0;
                if (in.is_open()) {
                        while (!in.eof()) {
                                in.read((char*)&(*tmp), sizeof(notion));
                                if (tmp->FIO == tmpPrevKey) return count;
                                addNode(tmp->FIO, count++);
                                tmpPrevKey = tmp->FIO;
                        }
                }
                in.close();
                lastRowNumber = count + 1;
                return count;
        }

        void showTree() {
                base->print();
        }

        void deleteNode(string FIO) {
                long long int iKey = convertFromFIOtoKey(FIO);
                if (iKey == -1) cout << "Too long fio, please cut it and try
again\n";
                else base->deleteNode(iKey);
        }

        void addNode(string FIO, int iRowNumber) {
                long long int iKey = convertFromFIOtoKey(FIO);
                if (iKey == -1) cout << "Too long fio, please cut it and try
again\n";
                else base->addNode(iKey, iRowNumber);
        }

        void findNode(string FIO) {
                int start_time = clock();
                auto start = chrono::high_resolution_clock::now();

                long long int iKey = convertFromFIOtoKey(FIO);
                if (iKey == -1)
                {
                        cout << "Too long fio, please cut it and try again\n";
                        return;
```

```
		}
		int iRowNumber = base->findNode(iKey);

		if (iRowNumber == -1) {
			cout << "It is not our student\n";
			return;
		}

		notion* tmp = new notion("", 0, false);
		ifstream fdirect(binaryFilePath, ios::in | ios::binary);
		fdirect.seekg((int)(iRowNumber) * sizeof(notion), ios::beg);
		fdirect.read((char*)&(*tmp), sizeof(notion));
		fdirect.close();

		cout << "\nStudent: " << tmp->FIO << " GPA: " << tmp->GPA << "
excluded status: " << tmp->excluded << endl;
		auto elapsed = chrono::high_resolution_clock::now() - start;
		long long microseconds =
std::chrono::duration_cast<std::chrono::microseconds>(elapsed).count();
		cout << "Execute time: " << microseconds << " microsec.\n";
		cout << "Count decisions: " << base->count << "\n";
	}

	long long int convertFromFIOtoKey(string stFIO) {
		if (stFIO.length() > 7) return -1;
		long long int result = 0;
		for (int i = stFIO.length() - 1; i >= 0 ; i--) {
			result += ((int)stFIO[i] - 32) * pow(100, stFIO.length() -
1 - i);
		}

		return result;
	}

	void generateFile(string stTextInputFile, int size) {
		ofstream out(stTextInputFile, ios::out);
		int* arr = new int[7];
		for (int i = 0; i < 7; i++) arr[i] = 2;
		for (int i = 0; i < size; i++) {
			notion* tmp = new notion(genearteString(7, arr),
((double)rand() / 100 + 5)*1.01, (bool)(rand()%3 - 1));
			out << tmp->FIO << " " << tmp->GPA << " " << tmp->excluded
<< endl;
		}
		out.close();
	}


	string genearteString(int size, int* arr) {
		string res = "";
		arr[6] += 1;
		for (int i = 1; i < size; i++) {
			if (arr[i] % 25 == 1) {
				arr[i - 1] += 1;
			}
		}
		for (int i = 0; i < size; i++) {
			res += (char)(arr[i]%25 + 97);
```

```
            }
            return res;
        }

};

#endif
```