Міністерство освіти і науки України
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"
Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Звіт

з лабораторної роботи № 1 з дисципліни
«Основи програмування 2. Модульне програмування»

«Бінарні файли»

Варіант 9

Виконав студент IП-15, Дзюбенко Даниїл Дмитрович
(шифр, прізвище, ім'я, по батькові)

Перевірив Вєчерковська Анастасія Сергіївна
(прізвище, ім'я, по батькові)

Київ 2022

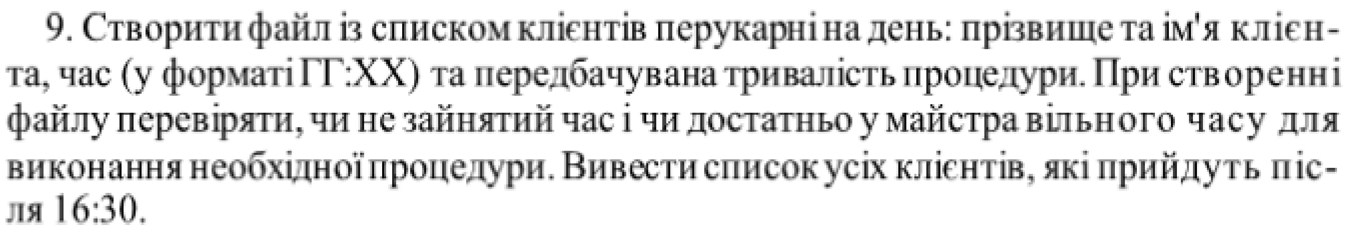
Лабораторна робота 1

Бінарні файли

Мета – вивчити особливості створення і обробки бінарних файлів даних.

Варіант 9

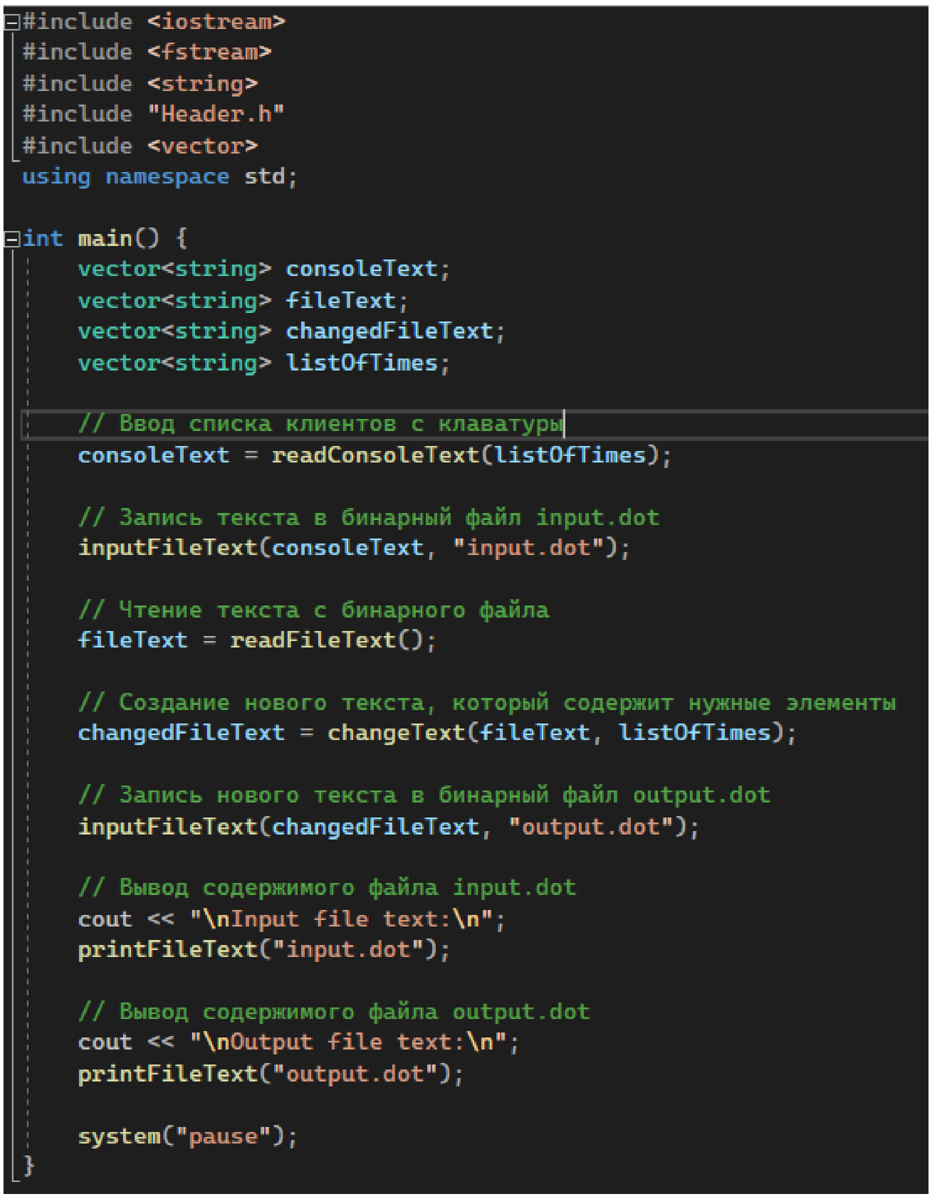
Задача



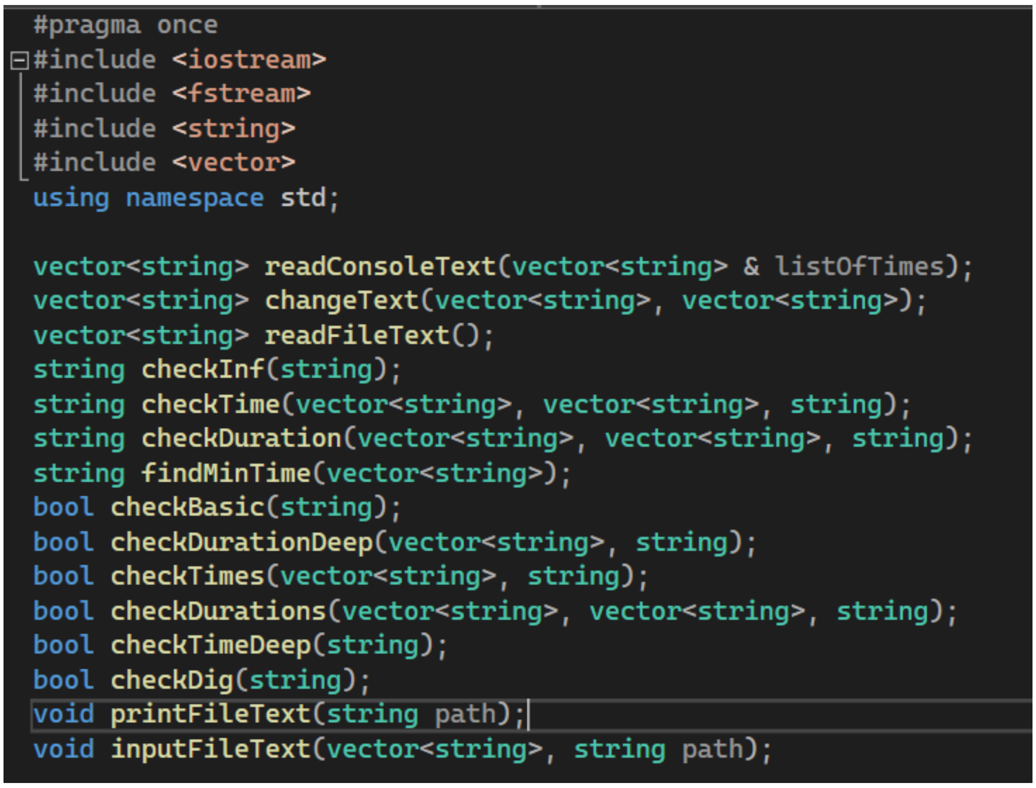
9. Створити файл із списком клієнтів перукарні на день: прізвище та ім'я клієнта, час (у форматі ГГ:ХХ) та передбачувана тривалість процедури. При створенні файлу перевіряти, чи не зайнятий час і чи достатньо у майстра вільного часу для виконання необхідної процедури. Вивести список усіх клієнтів, які прийдуть після 16:30.

Код

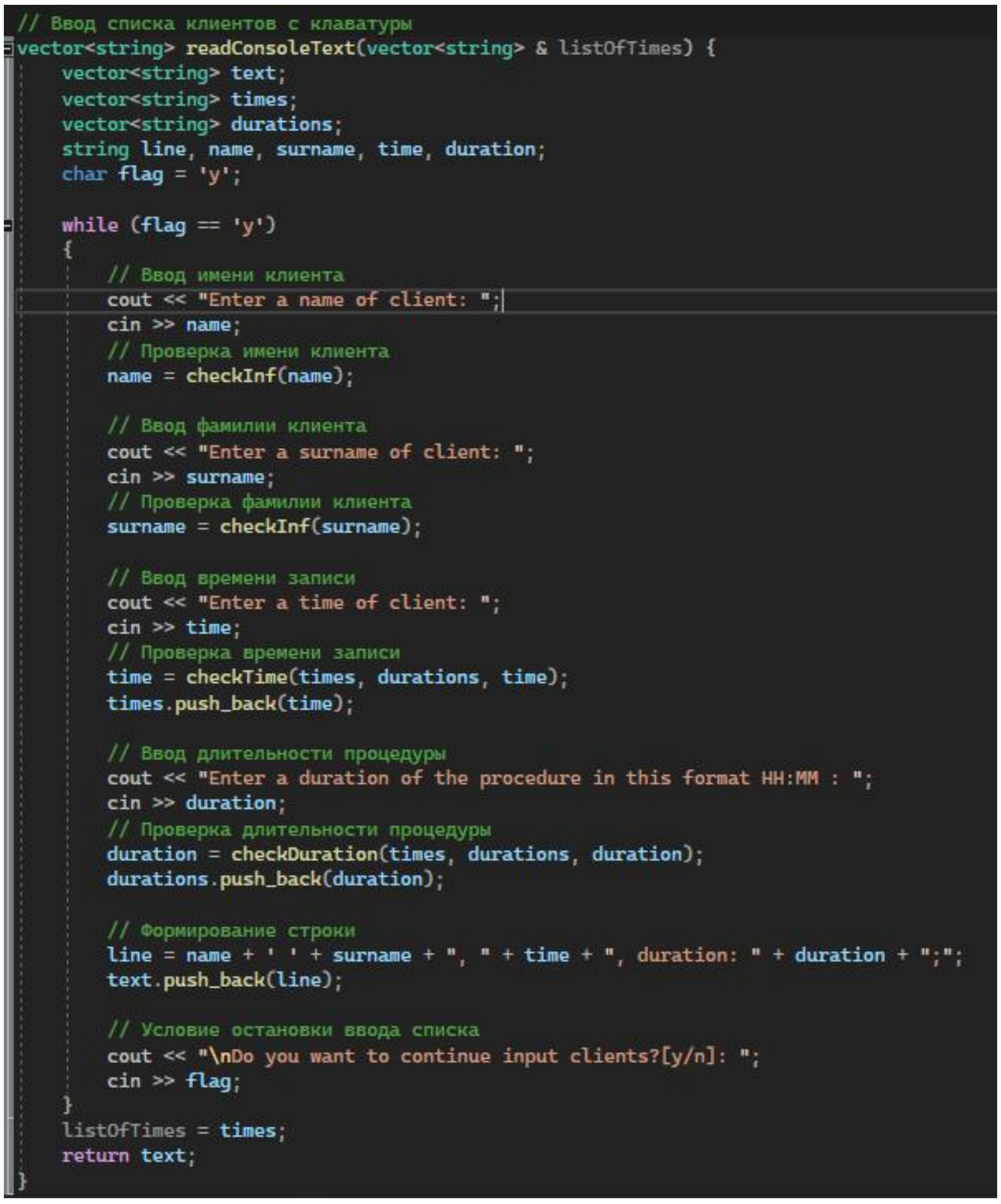
C++



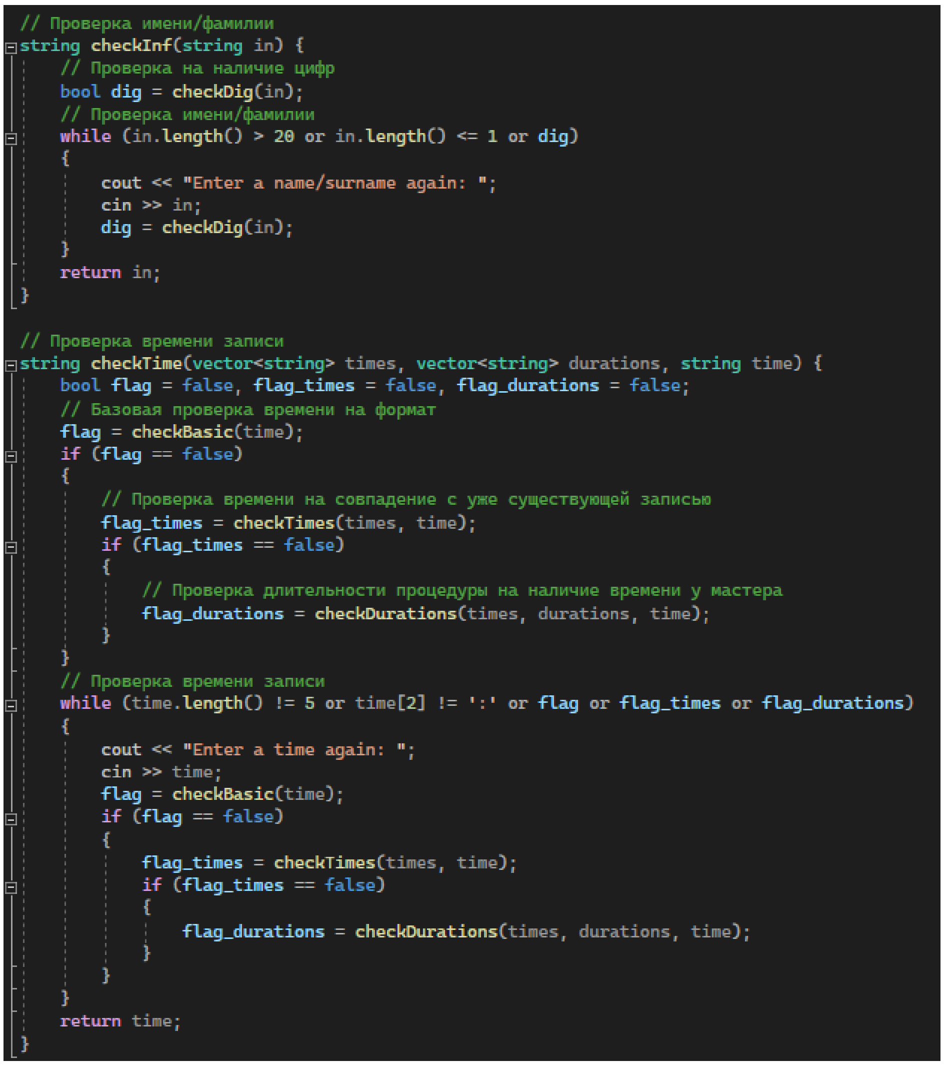
```cpp
#include <iostream>
#include <fstream>
#include <string>
#include "Header.h"
#include <vector>
using namespace std;

int main() {
    vector<string> consoleText;
    vector<string> fileText;
    vector<string> changedFileText;
    vector<string> listOfTimes;

    // Ввод списка клиентов с клаватуры
    consoleText = readConsoleText(listOfTimes);

    // Запись текста в бинарный файл input.dot
    inputFileText(consoleText, "input.dot");

    // Чтение текста с бинарного файла
    fileText = readFileText();

    // Создание нового текста, который содержит нужные элементы
    changedFileText = changeText(fileText, listOfTimes);

    // Запись нового текста в бинарный файл output.dot
    inputFileText(changedFileText, "output.dot");

    // Вывод содержимого файла input.dot
    cout << "\nInput file text:\n";
    printFileText("input.dot");

    // Вывод содержимого файла output.dot
    cout << "\nOutput file text:\n";
    printFileText("output.dot");

    system("pause");
}
```



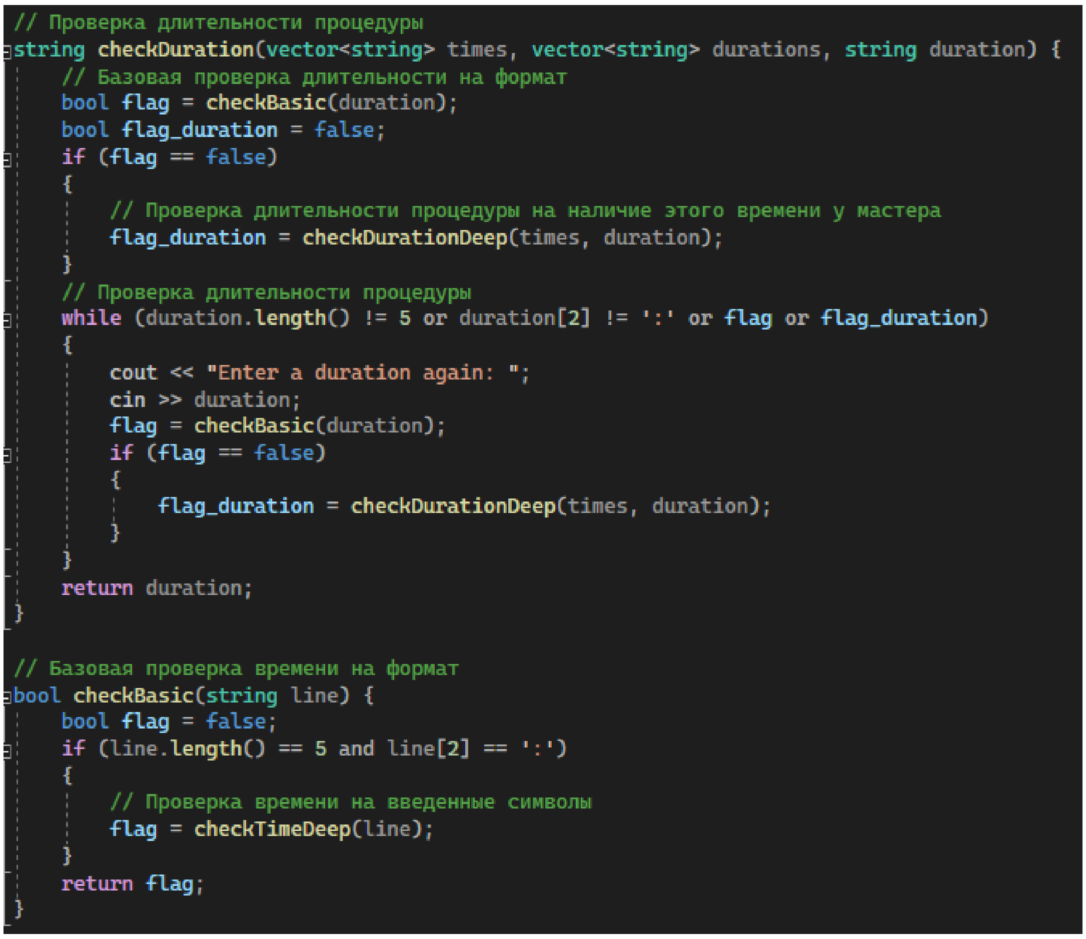
```cpp
#pragma once
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
using namespace std;

vector<string> readConsoleText(vector<string> & listOfTimes);
vector<string> changeText(vector<string>, vector<string>);
vector<string> readFileText();
string checkInf(string);
string checkTime(vector<string>, vector<string>, string);
string checkDuration(vector<string>, vector<string>, string);
string findMinTime(vector<string>);
bool checkBasic(string);
bool checkDurationDeep(vector<string>, string);
bool checkTimes(vector<string>, string);
bool checkDurations(vector<string>, vector<string>, string);
bool checkTimeDeep(string);
bool checkDig(string);
void printFileText(string path);
void inputFileText(vector<string>, string path);
```



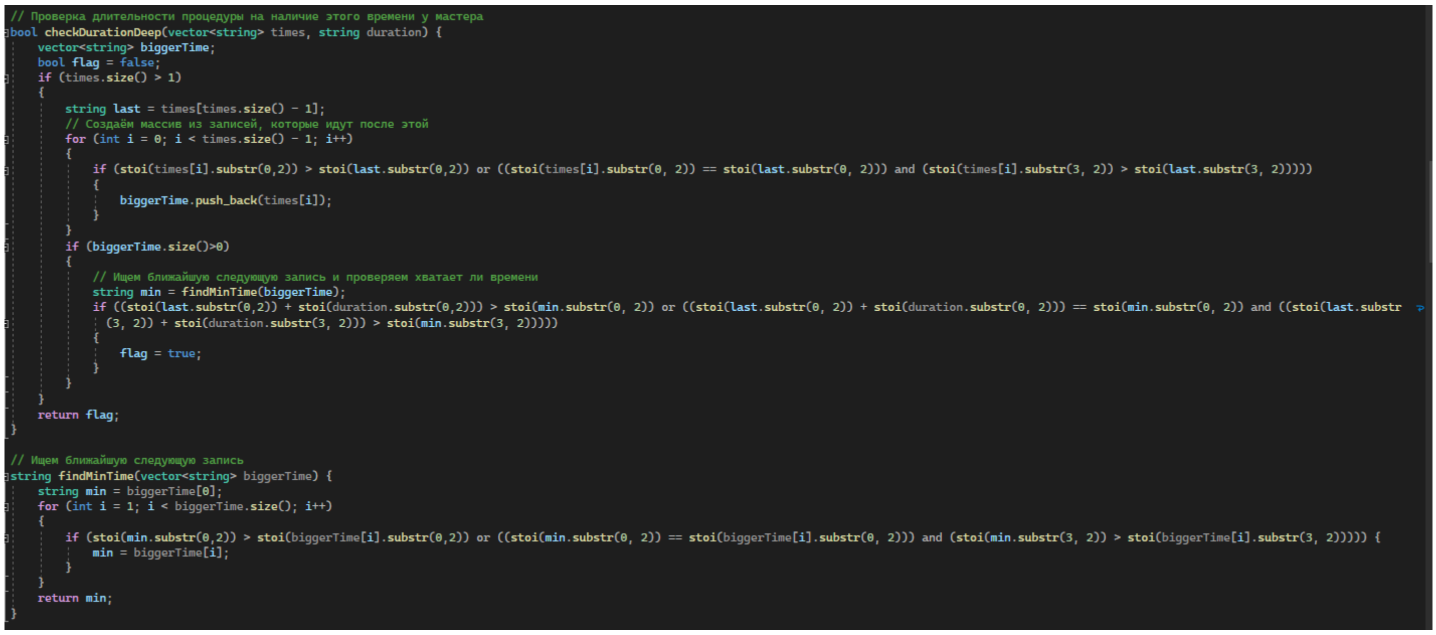
```cpp
// Ввод списка клиентов с клаватуры
vector<string> readConsoleText(vector<string> & listOfTimes) {
    vector<string> text;
    vector<string> times;
    vector<string> durations;
    string line, name, surname, time, duration;
    char flag = 'y';

    while (flag == 'y')
    {
        // Ввод имени клиента
        cout << "Enter a name of client: ";
        cin >> name;
        // Проверка имени клиента
        name = checkInf(name);

        // Ввод фамилии клиента
        cout << "Enter a surname of client: ";
        cin >> surname;
        // Проверка фамилии клиента
        surname = checkInf(surname);

        // Ввод времени записи
        cout << "Enter a time of client: ";
        cin >> time;
        // Проверка времени записи
        time = checkTime(times, durations, time);
        times.push_back(time);

        // Ввод длительности процедуры
        cout << "Enter a duration of the procedure in this format HH:MM : ";
        cin >> duration;
        // Проверка длительности процедуры
        duration = checkDuration(times, durations, duration);
        durations.push_back(duration);

        // Формирование строки
        line = name + ' ' + surname + ", " + time + ", duration: " + duration + ";";
        text.push_back(line);

        // Условие остановки ввода списка
        cout << "\nDo you want to continue input clients?[y/n]: ";
        cin >> flag;
    }
    listOfTimes = times;
    return text;
}
```



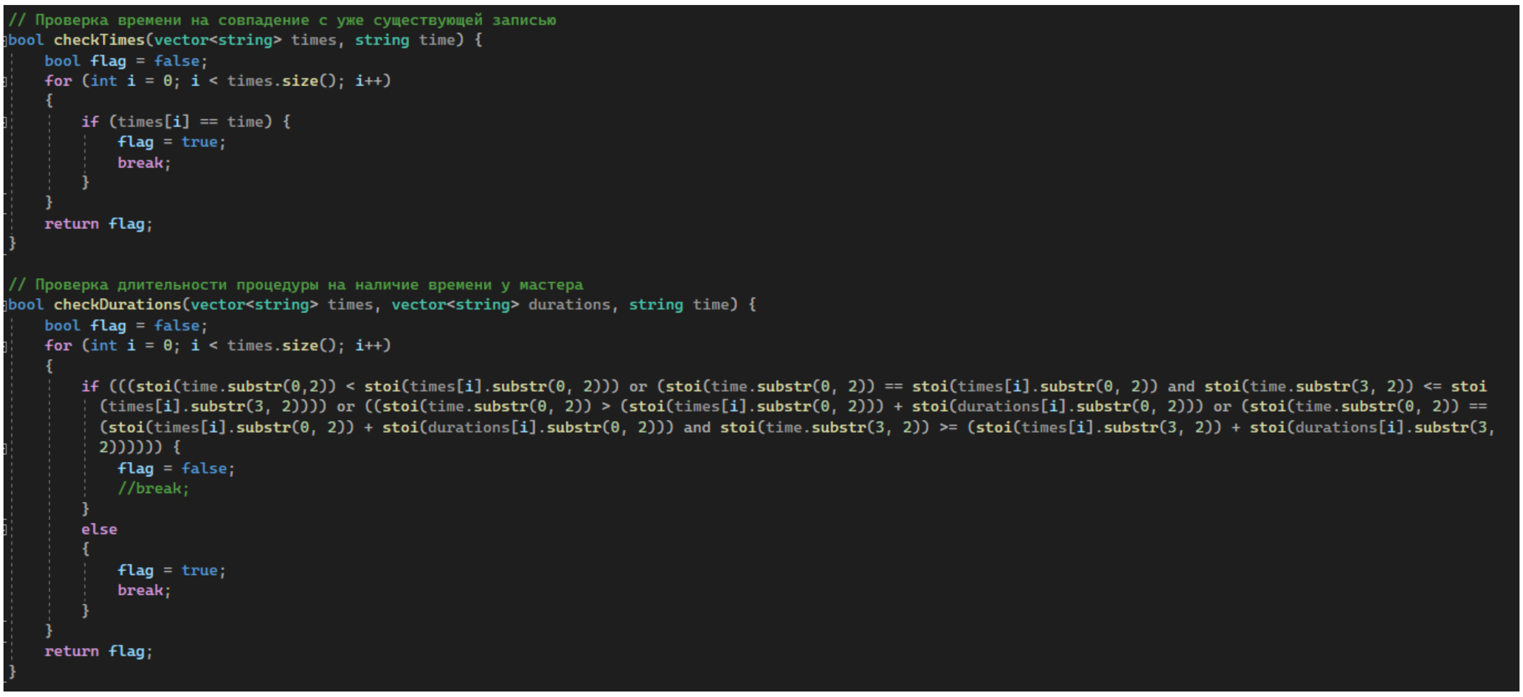
```
// Проверка имени/фамилии
string checkInf(string in) {
    // Проверка на наличие цифр
    bool dig = checkDig(in);
    // Проверка имени/фамилии
    while (in.length() > 20 or in.length() <= 1 or dig)
    {
        cout << "Enter a name/surname again: ";
        cin >> in;
        dig = checkDig(in);
    }
    return in;
}

// Проверка времени записи
string checkTime(vector<string> times, vector<string> durations, string time) {
    bool flag = false, flag_times = false, flag_durations = false;
    // Базовая проверка времени на формат
    flag = checkBasic(time);
    if (flag == false)
    {
        // Проверка времени на совпадение с уже существующей записью
        flag_times = checkTimes(times, time);
        if (flag_times == false)
        {
            // Проверка длительности процедуры на наличие времени у мастера
            flag_durations = checkDurations(times, durations, time);
        }
    }
    // Проверка времени записи
    while (time.length() != 5 or time[2] != ':' or flag or flag_times or flag_durations)
    {
        cout << "Enter a time again: ";
        cin >> time;
        flag = checkBasic(time);
        if (flag == false)
        {
            flag_times = checkTimes(times, time);
            if (flag_times == false)
            {
                flag_durations = checkDurations(times, durations, time);
            }
        }
    }
    return time;
}
```



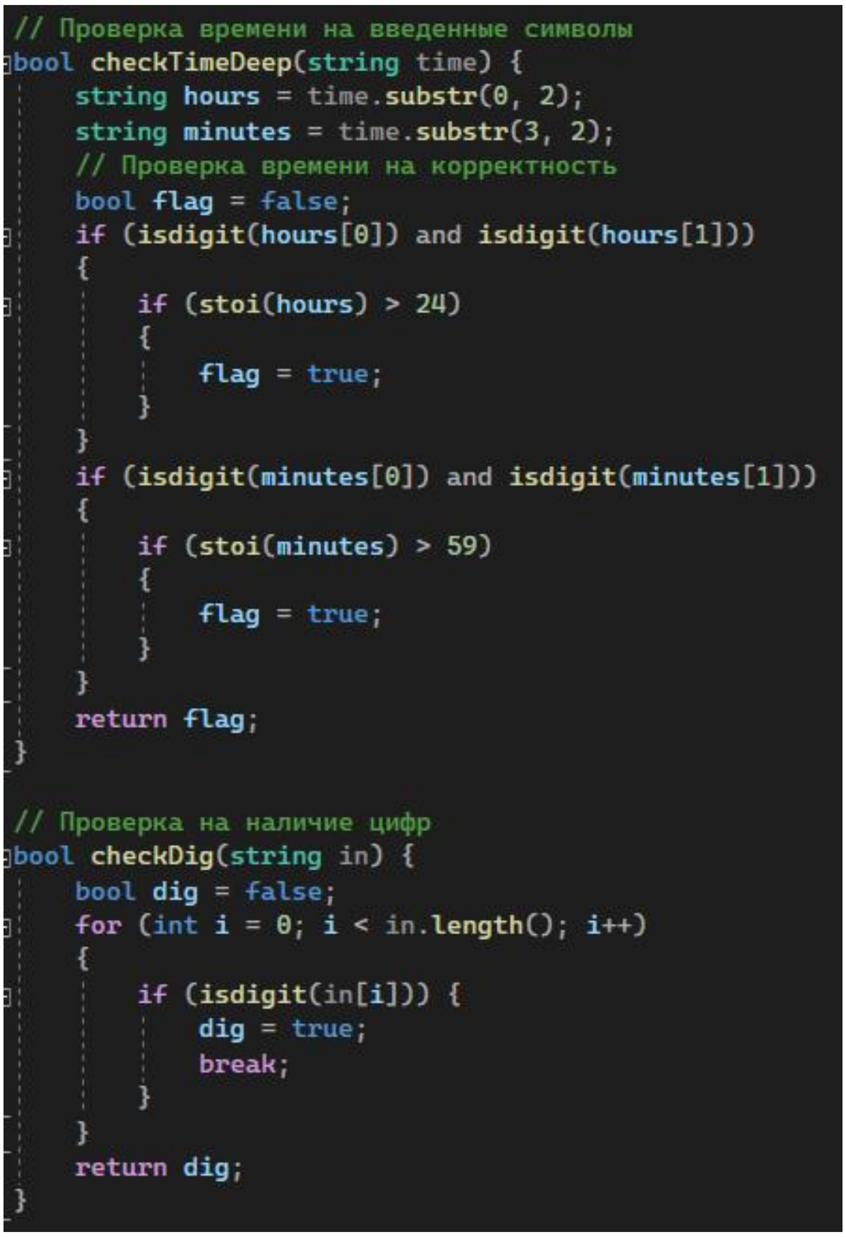
```
// Проверка длительности процедуры
string checkDuration(vector<string> times, vector<string> durations, string duration) {
    // Базовая проверка длительности на формат
    bool flag = checkBasic(duration);
    bool flag_duration = false;
    if (flag == false)
    {
        // Проверка длительности процедуры на наличие этого времени у мастера
        flag_duration = checkDurationDeep(times, duration);
    }
    // Проверка длительности процедуры
    while (duration.length() != 5 or duration[2] != ':' or flag or flag_duration)
    {
        cout << "Enter a duration again: ";
        cin >> duration;
        flag = checkBasic(duration);
        if (flag == false)
        {
            flag_duration = checkDurationDeep(times, duration);
        }
    }
    return duration;
}

// Базовая проверка времени на формат
bool checkBasic(string line) {
    bool flag = false;
    if (line.length() == 5 and line[2] == ':')
    {
        // Проверка времени на введенные символы
        flag = checkTimeDeep(line);
    }
    return flag;
}
```



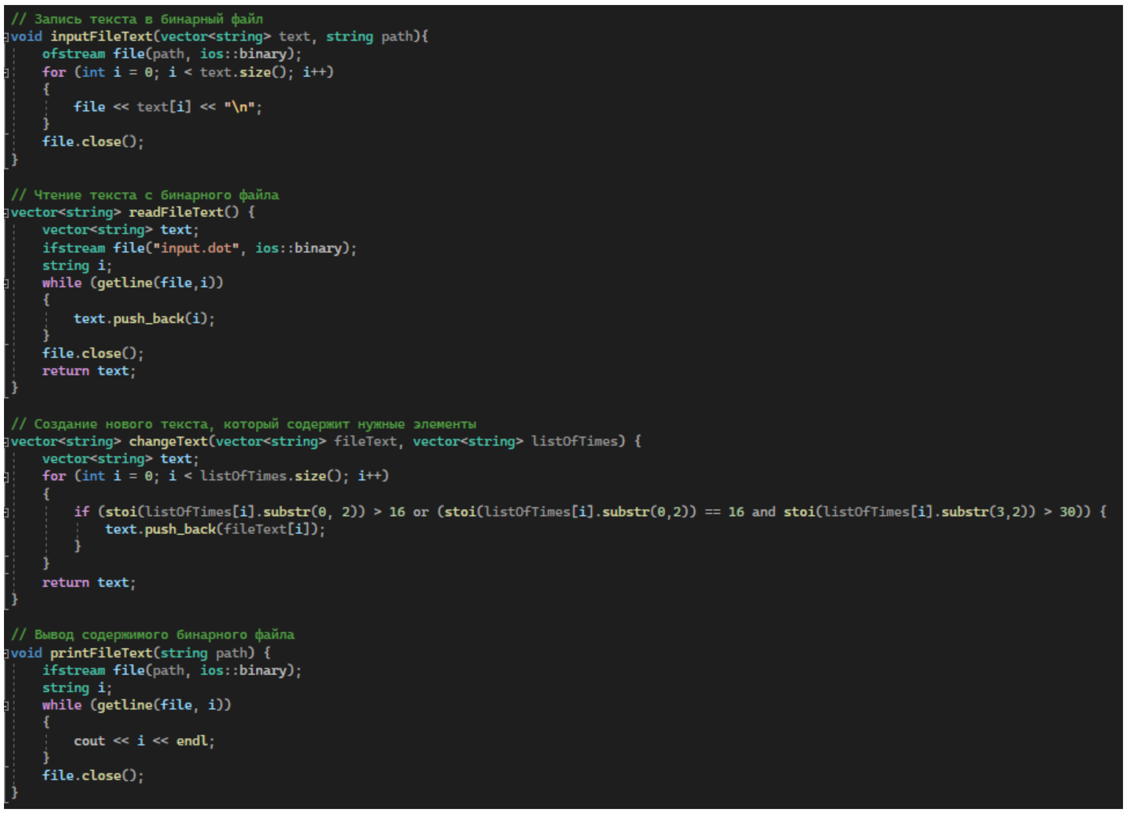
```cpp
// Проверка длительности процедуры на наличие этого времени у мастера
bool checkDurationDeep(vector<string> times, string duration) {
    vector<string> biggerTime;
    bool flag = false;
    if (times.size() > 1)
    {
        string last = times[times.size() - 1];
        // Создаём массив из записей, которые идут после этой
        for (int i = 0; i < times.size() - 1; i++)
        {
            if (stoi(times[i].substr(0,2)) > stoi(last.substr(0,2)) or ((stoi(times[i].substr(0, 2)) == stoi(last.substr(0, 2))) and (stoi(times[i].substr(3, 2)) > stoi(last.substr(3, 2)))))
            {
                biggerTime.push_back(times[i]);
            }
        }
        if (biggerTime.size()>0)
        {
            // Ищем ближайшую следующую запись и проверяем хватает ли времени
            string min = findMinTime(biggerTime);
            if ((stoi(last.substr(0,2)) + stoi(duration.substr(0,2))) > stoi(min.substr(0, 2)) or ((stoi(last.substr(0, 2)) + stoi(duration.substr(0, 2))) == stoi(min.substr(0, 2)) and ((stoi(last.substr
              (3, 2)) + stoi(duration.substr(3, 2))) > stoi(min.substr(3, 2)))))
            {
                flag = true;
            }
        }
    }
    return flag;
}

// Ищем ближайшую следующую запись
string findMinTime(vector<string> biggerTime) {
    string min = biggerTime[0];
    for (int i = 1; i < biggerTime.size(); i++)
    {
        if (stoi(min.substr(0,2)) > stoi(biggerTime[i].substr(0,2)) or ((stoi(min.substr(0, 2)) == stoi(biggerTime[i].substr(0, 2))) and (stoi(min.substr(3, 2)) > stoi(biggerTime[i].substr(3, 2))))) {
            min = biggerTime[i];
        }
    }
    return min;
}
```



```cpp
// Проверка времени на совпадение с уже существующей записью
bool checkTimes(vector<string> times, string time) {
    bool flag = false;
    for (int i = 0; i < times.size(); i++)
    {
        if (times[i] == time) {
            flag = true;
            break;
        }
    }
    return flag;
}

// Проверка длительности процедуры на наличие времени у мастера
bool checkDurations(vector<string> times, vector<string> durations, string time) {
    bool flag = false;
    for (int i = 0; i < times.size(); i++)
    {
        if (((stoi(time.substr(0,2)) < stoi(times[i].substr(0, 2))) or (stoi(time.substr(0, 2)) == stoi(times[i].substr(0, 2)) and stoi(time.substr(3, 2)) <= stoi
          (times[i].substr(3, 2)))) or ((stoi(time.substr(0, 2)) > (stoi(times[i].substr(0, 2))) + stoi(durations[i].substr(0, 2))) or (stoi(time.substr(0, 2)) ==
          (stoi(times[i].substr(0, 2)) + stoi(durations[i].substr(0, 2))) and stoi(time.substr(3, 2)) >= (stoi(times[i].substr(3, 2)) + stoi(durations[i].substr(3,
          2)))))) {
            flag = false;
            //break;
        }
        else
        {
            flag = true;
            break;
        }
    }
    return flag;
}
```

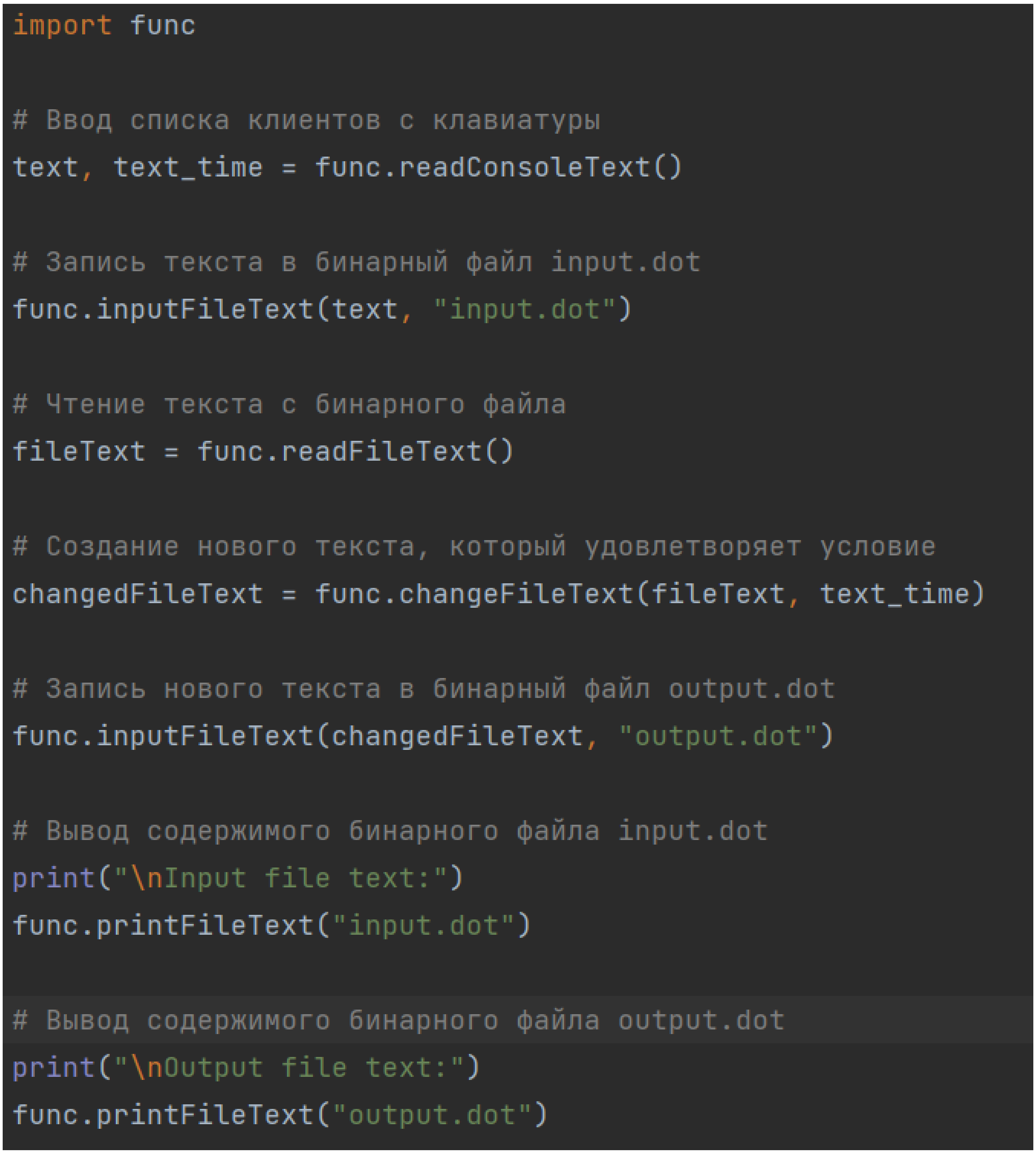


```cpp
// Проверка времени на введенные символы
bool checkTimeDeep(string time) {
    string hours = time.substr(0, 2);
    string minutes = time.substr(3, 2);
    // Проверка времени на корректность
    bool flag = false;
    if (isdigit(hours[0]) and isdigit(hours[1]))
    {
        if (stoi(hours) > 24)
        {
            flag = true;
        }
    }
    if (isdigit(minutes[0]) and isdigit(minutes[1]))
    {
        if (stoi(minutes) > 59)
        {
            flag = true;
        }
    }
    return flag;
}

// Проверка на наличие цифр
bool checkDig(string in) {
    bool dig = false;
    for (int i = 0; i < in.length(); i++)
    {
        if (isdigit(in[i])) {
            dig = true;
            break;
        }
    }
    return dig;
}
```

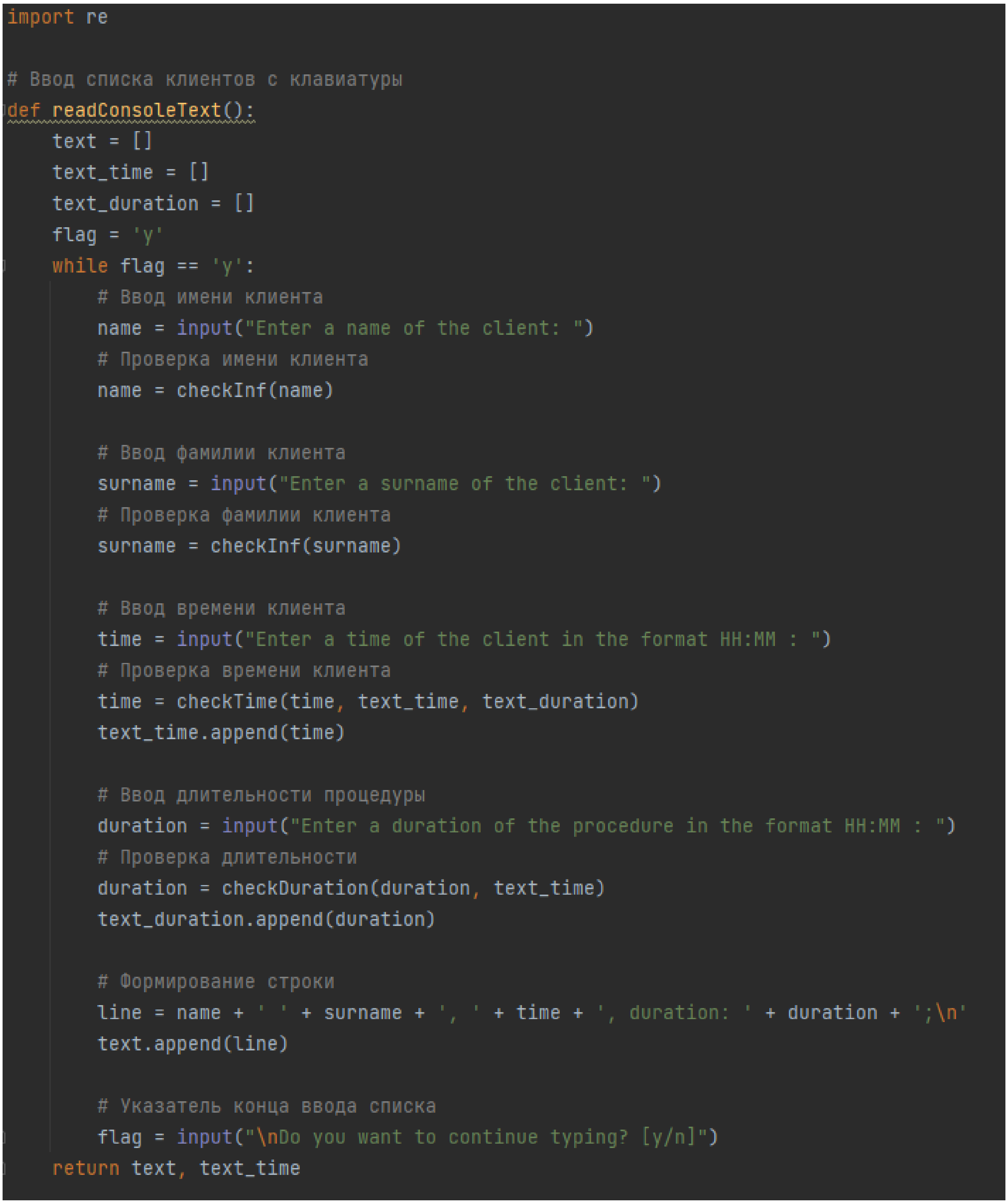


```cpp
// Запись текста в бинарный файл
void inputFileText(vector<string> text, string path){
    ofstream file(path, ios::binary);
    for (int i = 0; i < text.size(); i++)
    {
        file << text[i] << "\n";
    }
    file.close();
}

// Чтение текста с бинарного файла
vector<string> readFileText() {
    vector<string> text;
    ifstream file("input.dot", ios::binary);
    string i;
    while (getline(file,i))
    {
        text.push_back(i);
    }
    file.close();
    return text;
}

// Создание нового текста, который содержит нужные элементы
vector<string> changeText(vector<string> fileText, vector<string> listOfTimes) {
    vector<string> text;
    for (int i = 0; i < listOfTimes.size(); i++)
    {
        if (stoi(listOfTimes[i].substr(0, 2)) > 16 or (stoi(listOfTimes[i].substr(0,2)) == 16 and stoi(listOfTimes[i].substr(3,2)) > 30)) {
            text.push_back(fileText[i]);
        }
    }
    return text;
}

// Вывод содержимого бинарного файла
void printFileText(string path) {
    ifstream file(path, ios::binary);
    string i;
    while (getline(file, i))
    {
        cout << i << endl;
    }
    file.close();
}
```

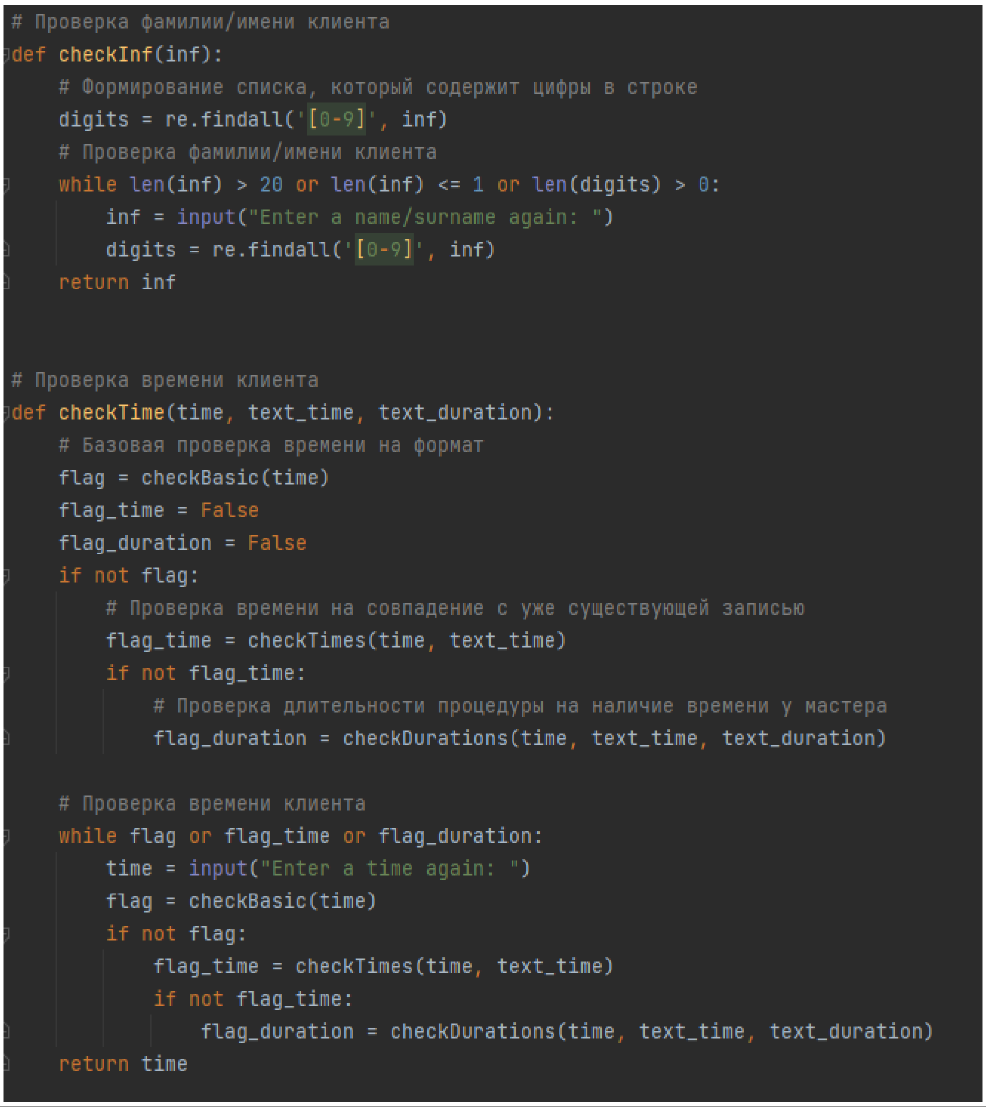
Python



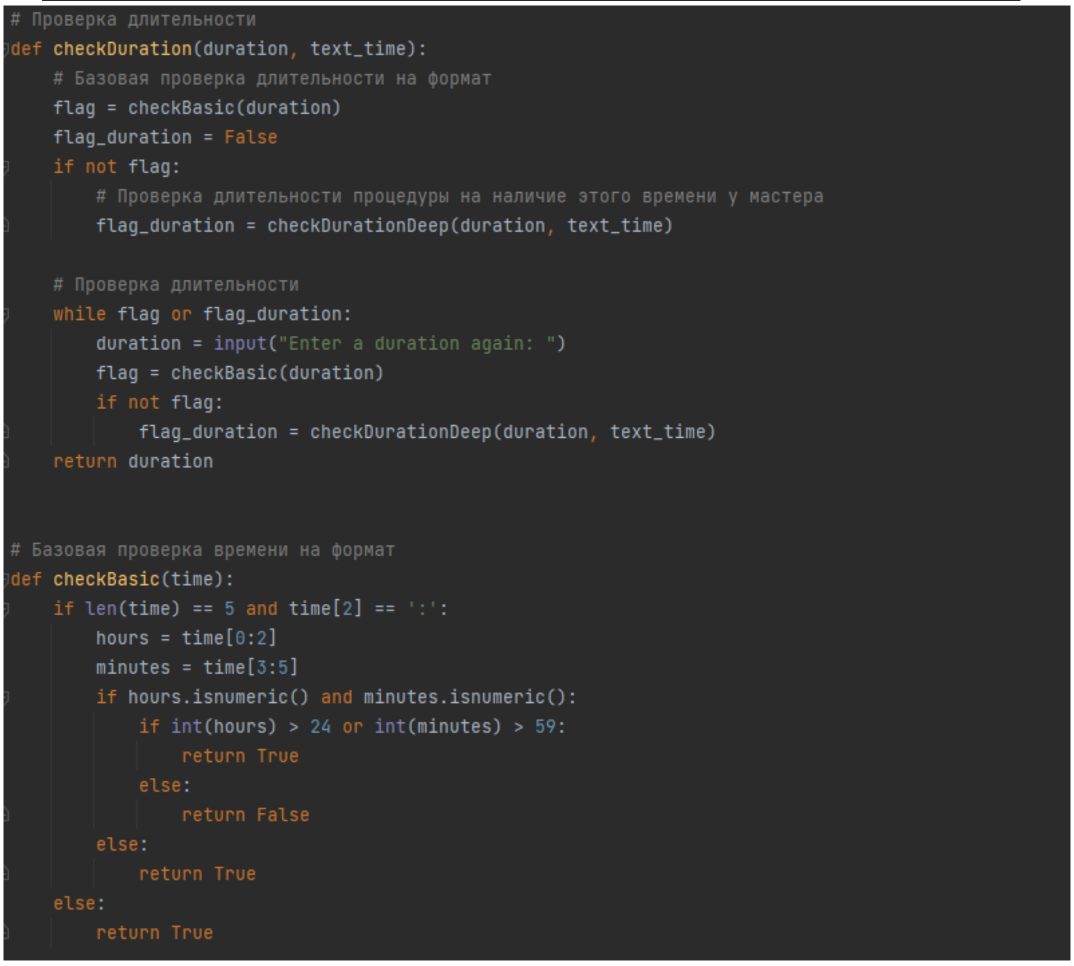
```python
import func

# Ввод списка клиентов с клавиатуры
text, text_time = func.readConsoleText()

# Запись текста в бинарный файл input.dot
func.inputFileText(text, "input.dot")

# Чтение текста с бинарного файла
fileText = func.readFileText()

# Создание нового текста, который удовлетворяет условие
changedFileText = func.changeFileText(fileText, text_time)

# Запись нового текста в бинарный файл output.dot
func.inputFileText(changedFileText, "output.dot")

# Вывод содержимого бинарного файла input.dot
print("\nInput file text:")
func.printFileText("input.dot")

# Вывод содержимого бинарного файла output.dot
print("\nOutput file text:")
func.printFileText("output.dot")
```



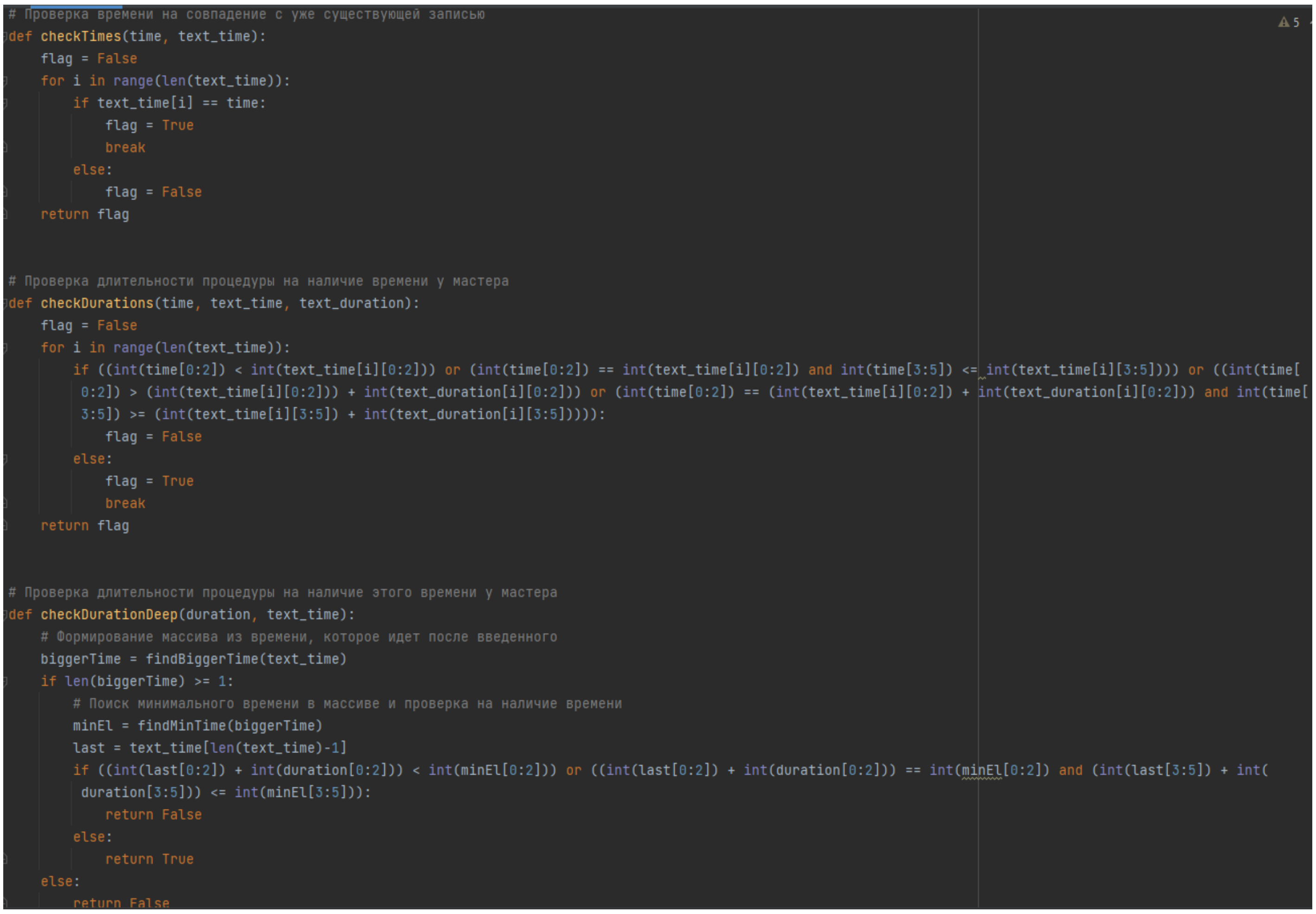
```python
import re

# Ввод списка клиентов с клавиатуры
def readConsoleText():
    text = []
    text_time = []
    text_duration = []
    flag = 'y'
    while flag == 'y':
        # Ввод имени клиента
        name = input("Enter a name of the client: ")
        # Проверка имени клиента
        name = checkInf(name)

        # Ввод фамилии клиента
        surname = input("Enter a surname of the client: ")
        # Проверка фамилии клиента
        surname = checkInf(surname)

        # Ввод времени клиента
        time = input("Enter a time of the client in the format HH:MM : ")
        # Проверка времени клиента
        time = checkTime(time, text_time, text_duration)
        text_time.append(time)

        # Ввод длительности процедуры
        duration = input("Enter a duration of the procedure in the format HH:MM : ")
        # Проверка длительности
        duration = checkDuration(duration, text_time)
        text_duration.append(duration)

        # Формирование строки
        line = name + ' ' + surname + ', ' + time + ', duration: ' + duration + ';\n'
        text.append(line)

        # Указатель конца ввода списка
        flag = input("\nDo you want to continue typing? [y/n]")
    return text, text_time
```



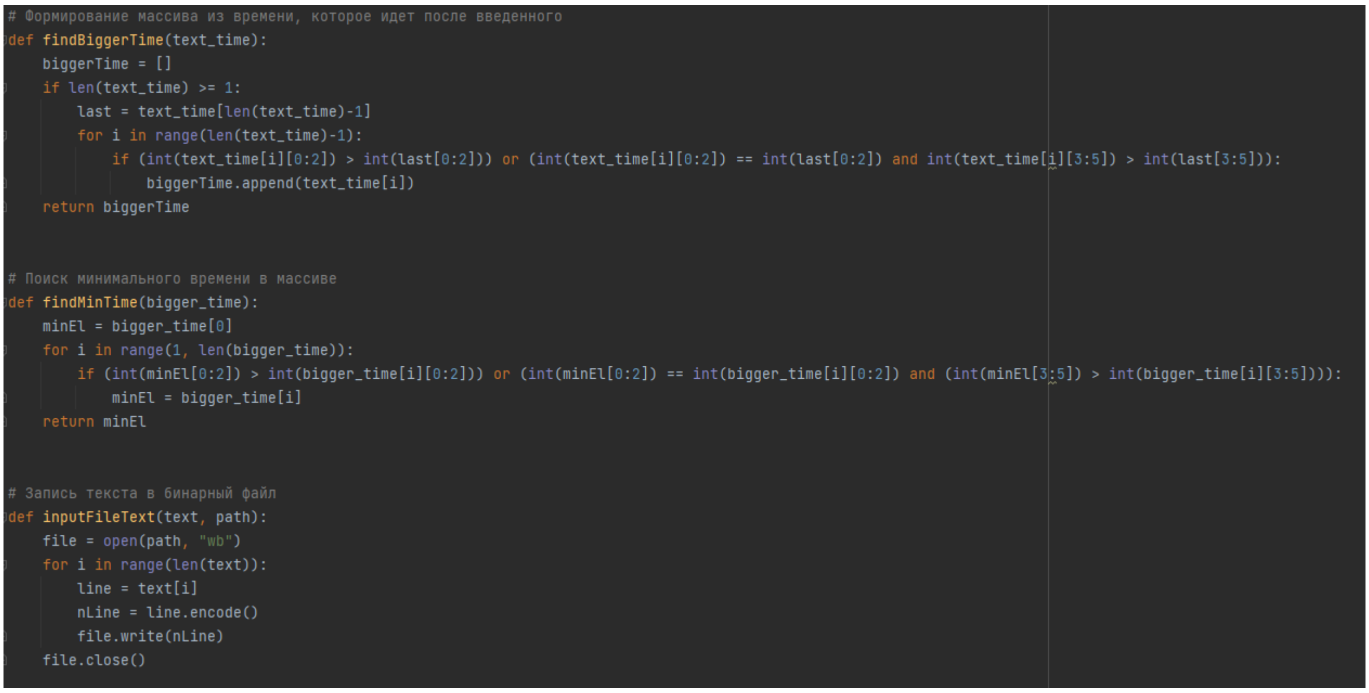
```python
# Проверка фамилии/имени клиента
def checkInf(inf):
    # Формирование списка, который содержит цифры в строке
    digits = re.findall('[0-9]', inf)
    # Проверка фамилии/имени клиента
    while len(inf) > 20 or len(inf) <= 1 or len(digits) > 0:
        inf = input("Enter a name/surname again: ")
        digits = re.findall('[0-9]', inf)
    return inf


# Проверка времени клиента
def checkTime(time, text_time, text_duration):
    # Базовая проверка времени на формат
    flag = checkBasic(time)
    flag_time = False
    flag_duration = False
    if not flag:
        # Проверка времени на совпадение с уже существующей записью
        flag_time = checkTimes(time, text_time)
        if not flag_time:
            # Проверка длительности процедуры на наличие времени у мастера
            flag_duration = checkDurations(time, text_time, text_duration)

    # Проверка времени клиента
    while flag or flag_time or flag_duration:
        time = input("Enter a time again: ")
        flag = checkBasic(time)
        if not flag:
            flag_time = checkTimes(time, text_time)
            if not flag_time:
                flag_duration = checkDurations(time, text_time, text_duration)
    return time
```



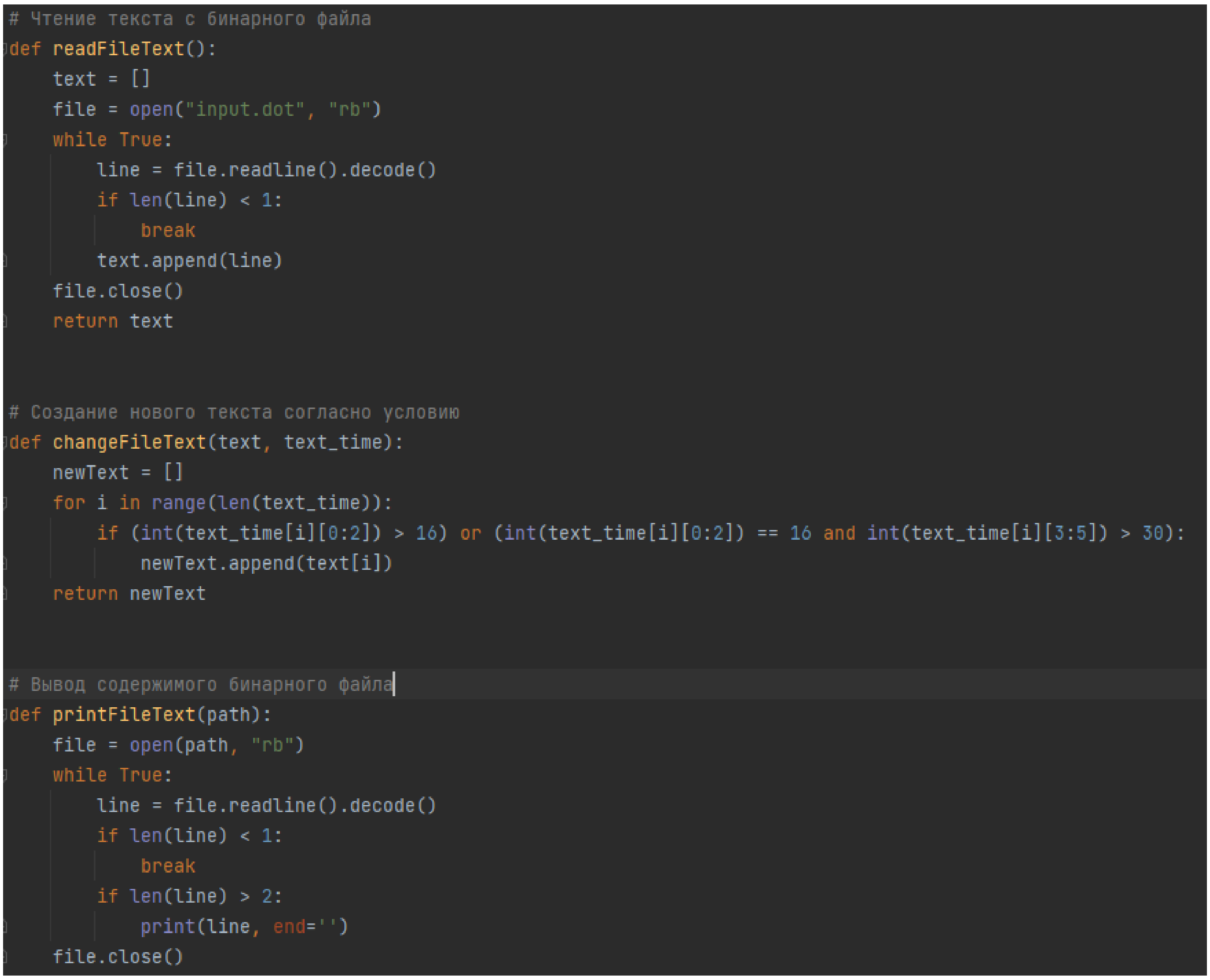
```python
# Проверка длительности
def checkDuration(duration, text_time):
    # Базовая проверка длительности на формат
    flag = checkBasic(duration)
    flag_duration = False
    if not flag:
        # Проверка длительности процедуры на наличие этого времени у мастера
        flag_duration = checkDurationDeep(duration, text_time)

    # Проверка длительности
    while flag or flag_duration:
        duration = input("Enter a duration again: ")
        flag = checkBasic(duration)
        if not flag:
            flag_duration = checkDurationDeep(duration, text_time)
    return duration


# Базовая проверка времени на формат
def checkBasic(time):
    if len(time) == 5 and time[2] == ':':
        hours = time[0:2]
        minutes = time[3:5]
        if hours.isnumeric() and minutes.isnumeric():
            if int(hours) > 24 or int(minutes) > 59:
                return True
            else:
                return False
        else:
            return True
    else:
        return True
```



```python
# Проверка времени на совпадение с уже существующей записью
def checkTimes(time, text_time):
    flag = False
    for i in range(len(text_time)):
        if text_time[i] == time:
            flag = True
            break
        else:
            flag = False
    return flag


# Проверка длительности процедуры на наличие времени у мастера
def checkDurations(time, text_time, text_duration):
    flag = False
    for i in range(len(text_time)):
        if ((int(time[0:2]) < int(text_time[i][0:2])) or (int(time[0:2]) == int(text_time[i][0:2]) and int(time[3:5]) <= int(text_time[i][3:5]))) or ((int(time[
         0:2]) > (int(text_time[i][0:2])) + int(text_duration[i][0:2])) or (int(time[0:2]) == (int(text_time[i][0:2]) + int(text_duration[i][0:2])) and int(time[
         3:5]) >= (int(text_time[i][3:5]) + int(text_duration[i][3:5])))):
            flag = False
        else:
            flag = True
            break
    return flag


# Проверка длительности процедуры на наличие этого времени у мастера
def checkDurationDeep(duration, text_time):
    # Формирование массива из времени, которое идет после введенного
    biggerTime = findBiggerTime(text_time)
    if len(biggerTime) >= 1:
        # Поиск минимального времени в массиве и проверка на наличие времени
        minEl = findMinTime(biggerTime)
        last = text_time[len(text_time)-1]
        if ((int(last[0:2]) + int(duration[0:2])) < int(minEl[0:2])) or ((int(last[0:2]) + int(duration[0:2])) == int(minEl[0:2]) and (int(last[3:5]) + int(
         duration[3:5])) <= int(minEl[3:5])):
            return False
        else:
            return True
    else:
        return False
```



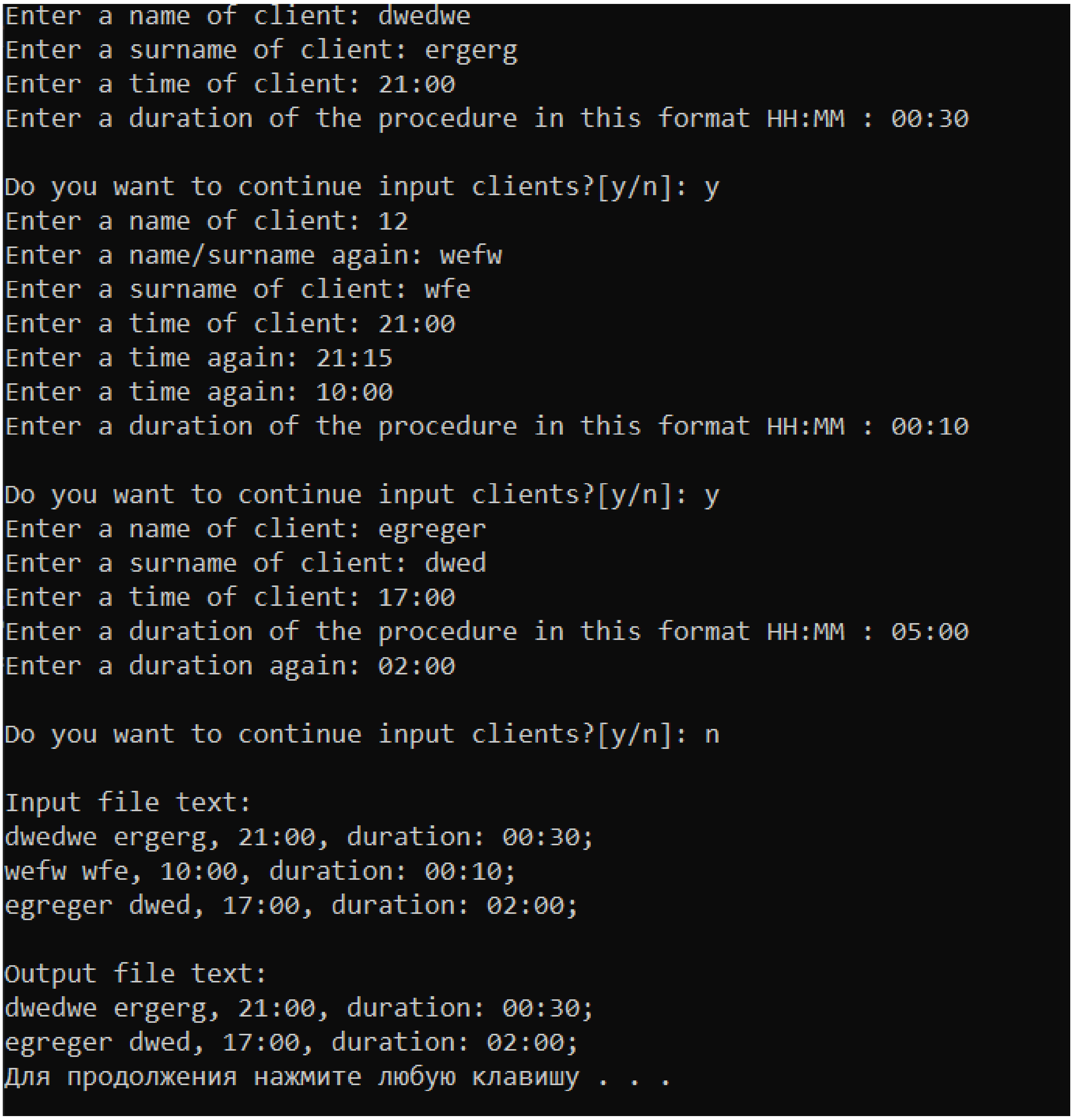
```python
# Формирование массива из времени, которое идет после введенного
def findBiggerTime(text_time):
    biggerTime = []
    if len(text_time) >= 1:
        last = text_time[len(text_time)-1]
        for i in range(len(text_time)-1):
            if (int(text_time[i][0:2]) > int(last[0:2])) or (int(text_time[i][0:2]) == int(last[0:2]) and int(text_time[i][3:5]) > int(last[3:5])):
                biggerTime.append(text_time[i])
    return biggerTime


# Поиск минимального времени в массиве
def findMinTime(bigger_time):
    minEl = bigger_time[0]
    for i in range(1, len(bigger_time)):
        if (int(minEl[0:2]) > int(bigger_time[i][0:2])) or (int(minEl[0:2]) == int(bigger_time[i][0:2]) and (int(minEl[3:5]) > int(bigger_time[i][3:5]))):
            minEl = bigger_time[i]
    return minEl


# Запись текста в бинарный файл
def inputFileText(text, path):
    file = open(path, "wb")
    for i in range(len(text)):
        line = text[i]
        nLine = line.encode()
        file.write(nLine)
    file.close()
```



```python
# Чтение текста с бинарного файла
def readFileText():
    text = []
    file = open("input.dot", "rb")
    while True:
        line = file.readline().decode()
        if len(line) < 1:
            break
        text.append(line)
    file.close()
    return text


# Создание нового текста согласно условию
def changeFileText(text, text_time):
    newText = []
    for i in range(len(text_time)):
        if (int(text_time[i][0:2]) > 16) or (int(text_time[i][0:2]) == 16 and int(text_time[i][3:5]) > 30):
            newText.append(text[i])
    return newText


# Вывод содержимого бинарного файла
def printFileText(path):
    file = open(path, "rb")
    while True:
        line = file.readline().decode()
        if len(line) < 1:
            break
        if len(line) > 2:
            print(line, end='')
    file.close()
```

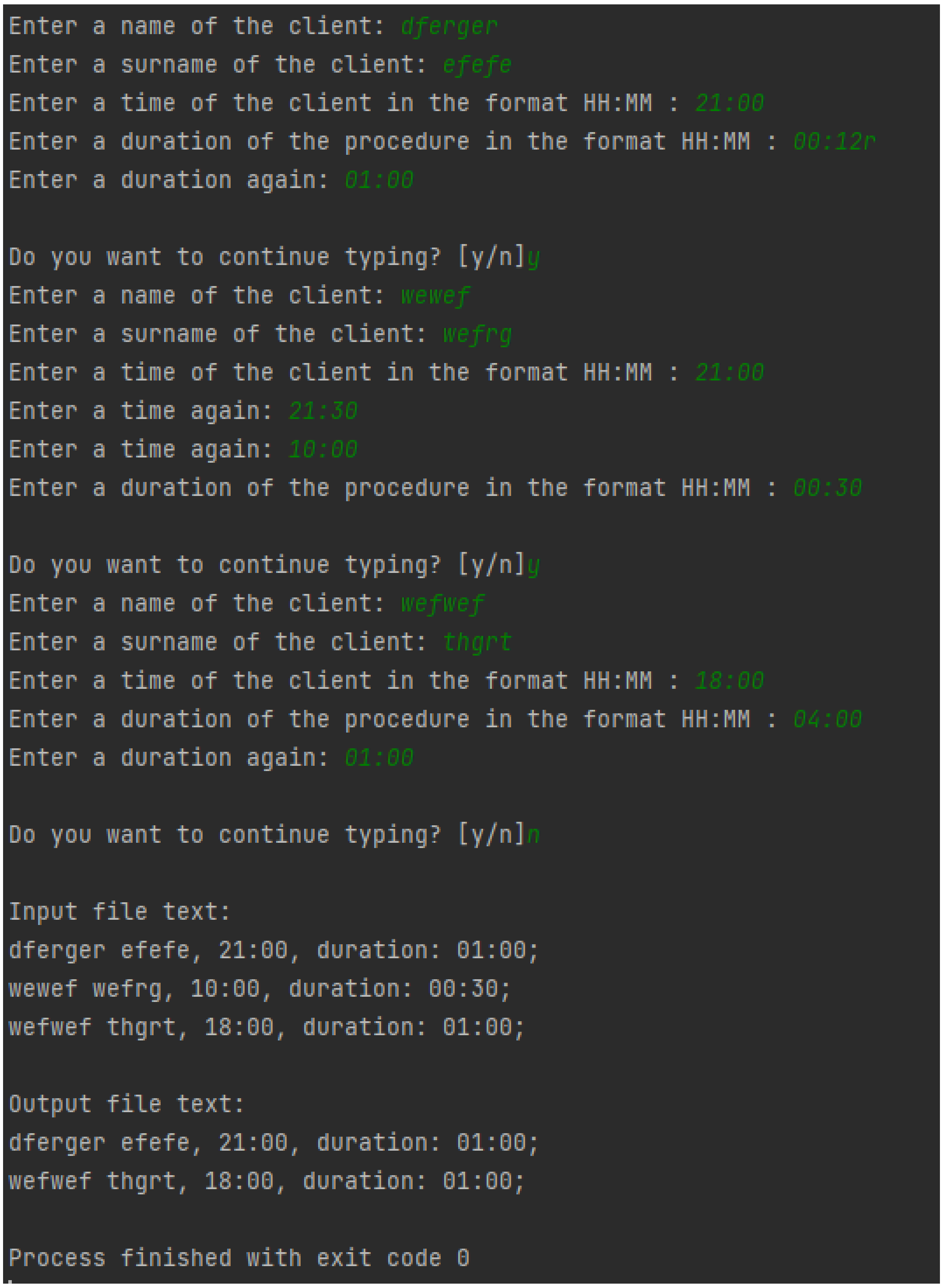
Тестування

C++



```
Enter a name of client: dwedwe
Enter a surname of client: ergerg
Enter a time of client: 21:00
Enter a duration of the procedure in this format HH:MM : 00:30

Do you want to continue input clients?[y/n]: y
Enter a name of client: 12
Enter a name/surname again: wefw
Enter a surname of client: wfe
Enter a time of client: 21:00
Enter a time again: 21:15
Enter a time again: 10:00
Enter a duration of the procedure in this format HH:MM : 00:10

Do you want to continue input clients?[y/n]: y
Enter a name of client: egreger
Enter a surname of client: dwed
Enter a time of client: 17:00
Enter a duration of the procedure in this format HH:MM : 05:00
Enter a duration again: 02:00

Do you want to continue input clients?[y/n]: n

Input file text:
dwedwe ergerg, 21:00, duration: 00:30;
wefw wfe, 10:00, duration: 00:10;
egreger dwed, 17:00, duration: 02:00;

Output file text:
dwedwe ergerg, 21:00, duration: 00:30;
egreger dwed, 17:00, duration: 02:00;
Для продолжения нажмите любую клавишу . . .
```

Python



```
Enter a name of the client: dferger
Enter a surname of the client: efefe
Enter a time of the client in the format HH:MM : 21:00
Enter a duration of the procedure in the format HH:MM : 00:12r
Enter a duration again: 01:00

Do you want to continue typing? [y/n]y
Enter a name of the client: wewef
Enter a surname of the client: wefrg
Enter a time of the client in the format HH:MM : 21:00
Enter a time again: 21:30
Enter a time again: 10:00
Enter a duration of the procedure in the format HH:MM : 00:30

Do you want to continue typing? [y/n]y
Enter a name of the client: wefwef
Enter a surname of the client: thgrt
Enter a time of the client in the format HH:MM : 18:00
Enter a duration of the procedure in the format HH:MM : 04:00
Enter a duration again: 01:00

Do you want to continue typing? [y/n]n

Input file text:
dferger efefe, 21:00, duration: 01:00;
wewef wefrg, 10:00, duration: 00:30;
wefwef thgrt, 18:00, duration: 01:00;

Output file text:
dferger efefe, 21:00, duration: 01:00;
wefwef thgrt, 18:00, duration: 01:00;

Process finished with exit code 0
```