**АКАДЕМИЯ УПРАВЛЕНИЯ ПРИ ПРЕЗИДЕНТЕ РЕСПУБЛИКИ БЕЛАРУСЬ**

ИНСТИТУТ УПРАВЛЕНЧЕСКИХ КАДРОВ

Факультет инновационной подготовки

Кафедра управления информационными ресурсами

Специальность «Управление информационными ресурсами»

**КУРСОВАЯ РАБОТА**

по дисциплине «Алгоритмизация и программирование»

на тему «Система учета рабочего времени сотрудников»

|  |
| --- |
| Студентка  2 курса, УИР-2 |
| Руководитель  Профессор |

Минск 2019

Э.Д. Фетисова

Н.И. Белодед

ОГЛАВЛЕНИЕ

[ВВЕДЕНИЕ 3](#_Toc7157728)

[ГЛАВА 1 ТЕОРИТИЧЕСКИЕ ОСНОВЫ ЯЗЫКА ПРОГРАММИРОВАНИЯ С++ 4](#_Toc7157729)

[ГЛАВА 2 СИСТЕМА УЧЕТА РАБОЧЕГО ВРЕМЕНИ СОТРУДНИКОВ 7](#_Toc7157731)

[2.1 Условие задачи 7](#_Toc7157732)

[2.2 Описание алгоритма (блок-схемы) программы 7](#_Toc7157734)

[2.2 Описание исходного кода программы 7](#_Toc7157735)

[2.2.1 Подключаемые библиотеки 7](#_Toc7157736)

[2.2.2 Описание файлов «Sotrudniki.h» и «Sotrudniki.cpp» 8](#_Toc7157737)

[2.2.3 Описание файлов «WorkTime.h» и «WorkTime.cpp» 10](#_Toc7157738)

[2.2.4 Описание файлов «SotrudnikManager.h» и «SotrudnikManager.cpp» 12](#_Toc7157739)

[2.2.5 Описание файлов «FileManager.h» и «FileManager.cpp» 17](#_Toc7157740)

[2.2.6 Описание файлов «Menu.h» и «Menu.cpp» 19](#_Toc7157741)

[2.2.7 Описание файла «Source.cpp» 20](#_Toc7157742)

[2.3 Результаты работы программы (скриншоты) 20](#_Toc7157743)

[ЗАКЛЮЧЕНИЕ 29](#_Toc7157744)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 30](#_Toc7157745)

[ПРИЛОЖЕНИЕ А 31](#_Toc7157746)

[ПРИЛОЖЕНИЕ Б 47](#_Toc7157747)

**ВВЕДЕНИЕ**

В настоящее время главным вопросом в любой организации является вопрос повышения эффективности работы самой организации. Одним из важнейших ресурсов любой организации является ее персонал. Время – это самый важный ресурс как для руководителя, так и для сотрудников.Учет рабочего времени сотрудников лежит в основе эффективного управления персоналом. Это необходимая мера для любой организации. В целях повышения эффективности работы персонала рекомендуется вести учет часов, отработанных каждым сотрудником.

Ведение такого учета позволяет значительно сократить потери временных трудозатрат и повысить уровень трудовой дисциплины работников организации.

Учет рабочего времени работников ведется в табеле рабочего времени. Табель применяется для контроля за соблюдением работниками установленного правилами внутреннего трудового распорядка, режима рабочего времени, получения сведений об отработанном времени каждым работником, для составления отчетности по труду.

С помощью этой системы есть возможность в любой момент времени получить информацию об отработанных каждым сотрудником часах с учетом всего времени пребывания на рабочем месте и перерывах. Получаемый отчет может включать в себя данные как по каждому из дней, так и итоговые за заданный период времени.

Объектом данной курсовой работы является система учета рабочего времени.

Предмет курсовой работы – информационная система учета рабочего времени в организации.

Таким образом, задачей курсовой работы является разработка консольного приложения на языке программирования высокого уровня C++ в среде Microsoft Visual Studio 2017, позволяющего оперировать базой данных сотрудников организации, вести табель учета рабочего времени.

**ГЛАВА 1  
ТЕОРИТИЧЕСКИЕ ОСНОВЫ ЯЗЫКА   
ПРОГРАММИРОВАНИЯ С++**

Язык программирования С++ был создан в начале 1980-х годов, его создатель сотрудник фирмы Bell Laboratories – Бьёрн Страуструп.

Объектно-ориентированное программирование – это парадигма программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса.

Одно из главных отличий языка С++ от С – это использование классов.

Используя объектно-ориентированный язык С++, мы можем определять свои собственные типы или классы. Класс представляет составной тип, который может использовать другие типы. Класс предназначен для описания некоторого типа объектов. То есть, по сути класс является планом объекта. А объект представляет конкретное воплощение класса, его реализацию.

*Объект* — это сущность, экземпляр класса, которой можно посылать сообщения, и которая может на них реагировать, используя свои данные. Данные объекта скрыты от остальной программы. Сокрытие данных называется инкапсуляцией.

*Инкапсуляция* — это свойство системы, позволяющее объединить данные и методы, работающие с ними в классе, и скрыть детали реализации от пользователя.

Ещё одним определяющим свойством для ООП является наследование.

*Наследование* — это свойство системы, позволяющее описать новый класс на основе уже существующего с частично или полностью заимствующейся функциональностью. Класс, от которого производится наследование, называется базовым, родительским или суперклассом. Новый класс — потомком, наследником или производным классом.

Наследование является важным, поскольку оно позволяет поддерживать концепцию иерархии классов. Применение иерархии классов делает управляемыми большие потоки информации.

Но даже наличие инкапсуляции и наследования не делает язык программирования в полной мере объектным с точки зрения ООП. Основные преимущества ООП проявляются только в том случае, когда в языке программирования реализован полиморфизм; то есть возможность объектов с одинаковой спецификацией иметь различную реализацию.

*Полиморфизм* — это свойство системы использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта.

Классы состоят из свойств и методов. Свойства – это любые данные, которыми можно характеризовать объект класса. Методы – это функции, которые могут выполнять какие-либо действия над данными (свойствами) класса.

Все свойства и методы классов имеют права доступа. По умолчанию, все содержимое класса является доступным для чтения и записи только для него самого. Для того, чтобы разрешить доступ к данным класса извне, используют модификатор доступа public. Все функции и переменные, которые находятся после модификатора public, становятся доступными из всех частей программы.

Закрытые данные класса размещаются после модификатора доступа private. Если отсутствует модификатор public, то все функции и переменные, по умолчанию являются закрытыми.

Обычно, приватными делают все свойства класса, а публичными – его методы.

Класс в объектно-ориентированном программировании может содержать конструкторы и деструкторы.

Конструкторы и деструкторы являются обычными функциями-членами или методами. Но они существуют в классе даже будучи не описанными пользователем явно.

*Конструктор* (от слова construct — создавать) – это специальный метод класса, который предназначен для инициализации элементов класса некоторыми начальными значениями.

В отличии от конструктора, деструктор (от слова destruct — разрушать) — специальный метод класса, который служит для уничтожения элементов класса. Чаще всего его используют тогда, когда в конструкторе, при создании объекта класса, динамически был выделен участок памяти и необходимо эту память очистить, если эти значения уже не нужны для дальнейшей работы программы.

Кроме вышеперечисленных, C++ предоставляет еще очень большое количество уникальных возможностей (использование виртуальных и встраиваемых функций, указатель this, операция расширенного доступа, шаблоны и исключения и так далее), что делает C++ одним из мощнейших на сегодняшний день объектно-ориентированных языков.

Таким образом, в данной главе были рассмотрены и проанализированы такие фундаментальные понятия объектно-ориентированного программирования, как полиморфизм, наследование, инкапсуляция, классы и объекты. Изучены свойства классов и их методы.

**ГЛАВА 2  
СИСТЕМА УЧЕТА РАБОЧЕГО ВРЕМЕНИ СОТРУДНИКОВ**

**2.1 Условие задачи**

Разработать систему учета рабочего времени сотрудников (Сведения о сотруднике, начало работы, окончание работы, перерывы, табель рабочего времени).

**2.2 Описание алгоритма (блок-схемы) программы**

Блок-схема представлены в ПРИЛОЖЕНИИ Б.

Использованные в блок-схеме элементы представлены в таблице 1.

**Таблица 1 – Элементы блок-схемы**

|  |  |  |
| --- | --- | --- |
| Название блока | Обозначение | Назначение блока |
| Терминатор |  | Начало, завершение программы или подпрограммы |
| Блок вычислений (определений) |  | Обработка данных (объявление/инициализация переменных, вычисления и так далее) |
| Логический блок |  | Ветвления, выбор, итерационные и поисковые циклы |
| Блок ввода/вывода |  | Ввод и вывод данных, чтение и запись данных в файл и тому подобное |

**2.2 Описание исходного кода программы**

Исходный код программы системы учета рабочего времени сотрудников представлен в ПРИЛОЖЕНИИ А.

**2.2.1 Подключаемые библиотеки**

Любая программа начинается с подключения библиотек.

В данной курсовой работе были использованы следующие библиотеки:

1. <iostream>
2. <fstream>
3. <string>
4. <Windows.h>

Заголовочный файл <iostream> управляет вводом/выводом. В данной курсовой работе использованы такие его объекты, как cin и cout , отвечающие за передачу информации в и из стандартных потоков ввода, вывода и ошибок без буферизации соответственно.

<fstream> позволяет реализовывать функции работы с файлами:

• Файловый ввод/вывод;

• open() – для открытия файла (в том числе в программе использованы режимы открытия файла ios\_base::app для записи в конец файла и ios\_base::trunc для предварительной очистки содержимого);

• close() – для закрытия файла;

• getline() – для считывания содержимого до конца строки;

• seekg() – для установления расстояния в байтах от начала файла до файлового указателя;

• eof() – для определения конца файла;

• remove() – для удаления файла;

• rename() – для изменения имени файла.

Заголовочный файл <string> определяет функци для обработки C-строк и массивов.

Заголовочный файл <Windows.h> используется для подключения русского языка в консоли SetConsoleCP, SetConsoleOutputCP.

**2.2.2 Описание файлов «Sotrudniki.h» и «Sotrudniki.cpp»**

В данном заголовочном файле осуществляется подключение стандартных библиотек С++ и описание класса Employee. В дальнейшем при подключении <Sotrudnik.h> в \*cpp-файлы, его содержимое будет автоматически включаться в \*cpp-файл, это необходимо делать для получения возможности пользоваться располагающимися в нём классами, полями и методами.

Для предотвращения переопределения используется директива #pragma с лексемой once, которая позволяет компилятору включить файл заголовка лишь однажды, независимо от того, сколько раз он был импортирован.

Класс Employee содержит переменные, спецификатора private, типа string: surname, name, patronymic, number, position – для хранения фамилии, имени, отчества, номера телефона и должности сотрудника и переменную типа int: employee\_code – для хранения кода сотрудника.

В файле «Sotrudnik.cpp» прописаны функции-члены класса Employee спецификатора public, конструктор с перегрузками и без, деструктор, а также функция:

void Employee::print\_employee() {

cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;

cout << "| Код сотрудника: | " << employee\_code << endl;

cout << "| ФИО сотрудника: | " << surname << " " << name << " " << patronymic << endl;

cout << "| Телефон сотрудника: | +" << number << endl;

cout << "| Должность сотрудника: | " << pozition << endl;

cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;

cout << endl;

}

– которая в отформатированном виде выводит информацию на экран. А также определены дружественные функции-операторы с перегрузкой:

friend ostream & operator<<(ostream & out, const Employee & obj) {

out << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;

out << "|Код сотрудника: | " << obj.employee\_code << endl;

out << "|Фамилия: | " << obj.surname << endl;

out << "|Имя: | " << obj.name << endl;

out << "|Отчество: | " << obj.patronymic << endl;

out << "|Телефон: | " << obj.number << endl;

out << "|Должность: | " << obj.pozition << endl;

out << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;

return out;

}

friend istream & operator>>(istream & in, Employee & obj) {

in >> obj.employee\_code;

in >> obj.surname;

in >> obj.name;

in >> obj.patronymic;

in >> obj.number;

in >> obj.pozition;

return in;

}

– дружественные функции, которые в отформатированном виде выводит информацию на экран из файла и загружает данные в файл.

**2.2.3 Описание файлов «WorkTime.h» и «WorkTime.cpp»**

В данном заголовочном файле осуществляется подключение стандартных библиотек С++ и описание класса WorkTime, объекты которого являются записями в табеле рабочего времени. В дальнейшем при подключении <WorkTime.h> в \*cpp-файлы, его содержимое будет автоматически включаться в \*cpp-файл.

Класс WorkTime содержит переменные типа int: empl\_code, StartWorkHour, StartWorkMin, FinishWorkHour, FinishWorkMin, StartBreakHour, FinishBreakHour, StartBreakMin, FinishBreakMin– для хранения кода сотрудника, начало и окончание рабочего времени (час и минуты), начало и окончание перерыва (час и минуты) и переменную типа string: date – для хранения времени.

В файле «WorkTime.cpp» прописаны функции-члены класса WorkTime, деструктор, конструктор с перегрузкой и без, а также функции:

int WorkTime::GetMinDiff() {

return ((FinishWorkHour - StartWorkHour) \* 60 + (FinishWorkMin - StartWorkMin))-((FinishBreakHour - StartBreakHour) \* 60 + (FinishBreakMin - StartBreakMin));

}

int WorkTime::GetHour() {

return GetMinDiff() / 60;

}

int WorkTime::GetMin() {

return GetMinDiff() % 60;

}

- которые считают время, отработанное сотрудниками, с учетом времени начала и окончания работы, а также время начала и окончания перерыва.

void WorkTime::print\_time() {

cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;

cout << "| Код сотрудника: | " << empl\_code << endl;

cout << "| Дата: | " << date << endl;

cout << "| Начало работы: | " << StartWorkHour << ":" << StartWorkMin << endl;

cout << "| Окончание работы: | " << FinishWorkHour << ":" << FinishWorkMin << endl;

cout << "| Начало перерыва: | " << StartBreakHour << ":" << StartBreakMin << endl;

cout << "| Окончание перерыва: | " << FinishBreakHour << ":" << FinishBreakMin << endl;

cout << "| Отработанное время за день: | " << GetHour() << " часов " << GetMin() << " минут" << endl;

cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;

cout << endl;

}

– которая в отформатированном виде выводит информацию на экран. А также определены дружественные функции-операторы с перегрузкой:

friend ostream & operator<<(ostream & outtime, const WorkTime & obj) {

outtime << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;

outtime << "|Код сотрудника: | " << obj.empl\_code << endl;

outtime << "|Дата: | " << obj.date << endl;

outtime << "|Начало работы: | " << obj.StartWorkHour << ":" << obj.StartWorkMin << endl;

outtime << "|Окончание работы: | " << obj.FinishWorkHour << ":" << obj.FinishWorkMin << endl;

outtime << "|Начало перерыва: | " << obj.StartBreakHour << ":" << obj.StartBreakMin<< endl;

outtime << "|Окончание перерыва: | " << obj.FinishBreakHour << ":" << obj.FinishBreakMin << endl;

outtime << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;

return outtime;

}

friend istream & operator>>(istream & intime, WorkTime & obj) {

intime >> obj.empl\_code;

intime >> obj.date;

intime >> obj.StartWorkHour;

intime >> obj.StartWorkMin;

intime >> obj.FinishWorkHour;

intime >> obj.FinishWorkMin;

intime >> obj.StartBreakHour;

intime >> obj.StartBreakMin;

intime >> obj.FinishBreakHour;

intime >> obj.FinishBreakMin;

return intime;

}

– которые в отформатированном виде выводит информацию на экран из файла и загружает данные в файл.

**2.2.4 Описание файлов «SotrudnikManager.h» и «SotrudnikManager.cpp»**

В файле исходного кода «SotrudnikManager.сpp» прописаны методы класса, управляющие записями табеля рабочего времени и базой данных сотрудников. Файл содержит массивы классов Employee и WorkTime.

Класс EmployeeManager наследуется от двух базовых классов Employee и WorkTime и наследует их поля, также содержит переменные типа int: currentEmployeeSize, maxEmployeeSize, currentTimeWorkSize, maxTimeWorkSize – для хранения данных о текущем и максимальном размерах для выделения динамической памяти для Employee и WorkTime.

В файле «SotrudnikManager.сpp» прописаны функции-члены классов Employee и WorkTime, а именно:

Конструктор:

EmployeeManager::EmployeeManager() {

currentEmployeeSize = 0;

maxEmployeeSize = SIZE;

employee = new Employee[maxEmployeeSize];

currentTimeWorkSize = 0;

maxTimeWorkSize = SIZE;

time = new WorkTime[maxTimeWorkSize];

}

- который служит для динамического выделения памяти для классов Employee и WorkTime.

EmployeeManager::EmployeeManager(int employee\_code, string surname, string name, string patronymic, string number, string pozition) : Employee(employee\_code, surname, name, patronymic, number, pozition) {}

EmployeeManager::EmployeeManager(int empl\_code, string date, int StartWorkHour, int StartWorkMin, int FinishWorkHour, int FinishWorkMin, int StartBreakHour, int StartBreakMin, int FinishBreakHour, int FinishBreakMin) : WorkTime(empl\_code, date, StartWorkHour, StartWorkMin, FinishWorkHour, FinishWorkMin, StartBreakHour, StartBreakMin, FinishBreakHour, FinishBreakMin) {}

- конструкторы, которым передаются параметры наследуемых классов.

И деструктор, который служит для освобождения памяти:

EmployeeManager::~EmployeeManager() {

delete[] employee;

delete[] time;

}

А также функции:

void EmployeeManager::add\_employee() {

int employee\_code;

string surname;

string name;

string patronymic;

string number;

string pozition;

cout << "Введите код сотрудника: ";

cin >> employee\_code;

cout << "Введите фамилию сотрудника: ";

cin >> surname;

cout << "Введите имя сотрудника: ";

cin >> name;

cout << "Введите отчество сотрудника: ";

cin >> patronymic;

cout << "Введите номер телефона сотрудника: +";

cin >> number;

cout << "Введите должность сотрудника: ";

cin >> pozition;

Employee sup(employee\_code, surname, name, patronymic, number, pozition);

addEmployee(sup);

}

– которая служит для ввода информации о сотруднике с клавиатуры

void EmployeeManager::add\_time() {

int StartWorkHour;

int StartWorkMin;

int FinishWorkHour;

int FinishWorkMin;

int StartBreakHour;

int StartBreakMin;

int FinishBreakHour;

int FinishBreakMin;

int empl\_code;

string date;

cout << "Код сотрудника: ";

cin >> empl\_code;

cout << "Введите дату в формате день.месяц.год: ";

cin >> date;

cout << "Введите время начала работы (час): ";

cin >> StartWorkHour;

cout << "Введите время начала работы (мин) : ";

cin >> StartWorkMin;

cout << "Введите время окончания работы (час) : ";

cin >> FinishWorkHour;

cout << "Введите время окончания работы (мин) : ";

cin >> FinishWorkMin;

cout << "Введите время начала перерыва (час): ";

cin >> StartBreakHour;

cout << "Введите время перерыва работы (мин) : ";

cin >> StartBreakMin;

cout << "Введите время окончания перерыва(час) : ";

cin >> FinishBreakHour;

cout << "Введите время окончания перерыва (мин) : ";

cin >> FinishBreakMin;

WorkTime tw(empl\_code, date, StartWorkHour, StartWorkMin, FinishWorkHour, FinishWorkMin, StartBreakHour, StartBreakMin, FinishBreakHour, FinishBreakMin);

addTime(tw);

}

– которая служит для добавления записи в табель рабочего времени с клавиатуры.

Функции добавления сотрудника и записи в табель:

void EmployeeManager::addEmployee(Employee empl) {

if (currentEmployeeSize == maxEmployeeSize) {

Employee\* temp = employee;

int newSize = 2 \* maxEmployeeSize + 1;

employee = new Employee[newSize];

for (int i = 0; i < maxEmployeeSize; i++) {

\*(employee + i) = \*(temp + i);

}

delete[] temp;

maxEmployeeSize = newSize;

}

employee[currentEmployeeSize++] = empl;

}

void EmployeeManager::addTime(WorkTime tw) {

if (currentTimeWorkSize == maxTimeWorkSize) {

WorkTime\* temp = time;

int newSize = 2 \* maxTimeWorkSize + 1;

time = new WorkTime[newSize];

for (int i = 0; i < maxTimeWorkSize; i++) {

\*(time + i) = \*(temp + i);

}

delete[] temp;

maxTimeWorkSize = newSize;

}

time[currentTimeWorkSize++] = tw;

}

- которые выделяют динамическую память под массив данных классов Employee и WorkTime.

void EmployeeManager::printEmployee() {

for (int i = 0; i < currentEmployeeSize; i++) {

employee[i].print\_employee();

}

}

EmployeeManager\* EmployeeManager::printTime() {

for (int i = 0; i < currentTimeWorkSize; i++) {

time[i].print\_time();

}

return nullptr;

}

Функции printEmployee() и printTime () выводят на экран информацию о сотрудниках из массивов, используя метод класса Employee – print\_employee()и WorkTime – print\_time().

А также функции удаления сотрудников:

// удалить всех сотрудников

void EmployeeManager::deleteAll() {

currentEmployeeSize = 0;

cout << "Все сотрудники успешно удалены" << endl;

}

// удалить сотрудника по фамилии

Employee\* EmployeeManager::deleteEmployeeBySurname(string surname) {

for (int i = 0; i < currentEmployeeSize; i++) {

if (employee[i].getsurname() == surname) {

for (int j = i; j < currentEmployeeSize - 1; j++) {

employee[j] = employee[j + 1];

}

currentEmployeeSize--;

}

}

return nullptr;

}

В этой функции происходит проверка на совпадение введенных данных пользователя и содержимого и удаляет, если они совпадают.

Функции поиска информации с параметром кода сотрудника:

// найти сотрудника по коду

Employee\* EmployeeManager::findEmployeeByCode(int employee\_code) {

for (int i = 0; i < currentEmployeeSize; i++) {

if (employee[i].getemployee\_code() == employee\_code) {

return &employee[i];

}

}

return nullptr;

}

Функция с параметром employee\_code выводит данные сотрудника по определенному коду сотрудника, заданным пользователем, сравнивая переданные значения переменных с кодом в текущей записи массива.

// найти рабочее время сотрудника по коду

WorkTime\* EmployeeManager::findWorkTimeByEmployeeCode(int employee\_code) {

for (int i = 0; i < currentTimeWorkSize; i++) {

if (time[i].getempl\_code() == employee\_code) {

time[i].print\_time();

}

}

return nullptr;

}

Функция с параметром employee\_code выводит данные табеля рабочего времени по определенному коду сотрудника, заданным пользователем, сравнивая переданные значения переменных с кодом в текущей записи массива.

**2.2.5 Описание файлов «FileManager.h» и «FileManager.cpp»**

В этом файле описаны функции записи данных файлы «sotrudniki.txt» и «time.txt»:

void FileManager::SaveToFile(EmployeeManager\* ems){

ofstream out("sotrudniki.txt");

if (out.is\_open()) {

int size = ems->getEmployeeCurrentSize();

out << size << endl;

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

out << ems->getEmployee()[i];

}

out.close();

}

else {

cout << "Ошибка открытия файла!" << endl;

}

ofstream outtime("WorkTime.txt");

if (outtime.is\_open()) {

int size = ems->getTimeCurrentSize();

outtime << size << endl;

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

outtime << ems->getTime()[i];

}

outtime.close();

}

else {

cout << "Ошибка открытия файла!" << endl;

}

}

Следующая функция загружает базу сотрудников и табель рабочего времени из файлов «sotrudniki.txt» и «time.txt»Перед загрузкой из файлов содержимое массивов очищается.

void FileManager::LoadFromFile(EmployeeManager\* ems){

ifstream in("sotrudniki.txt");

if (in.is\_open()) {

int size;

in >> size;

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

Employee empl;

in >> empl;

ems->addEmployee(empl);

while (in.read((char\*)&empl, sizeof(Employee))) {

empl.print\_employee();

}

}

in.close();

cout << "Данные сотрудников считаны\n" << endl;

}

else {

cout << "Ошибка открытия файла!" << endl;

}

ifstream intime("WorkTime.txt");

if (intime.is\_open()) {

int size;

intime >> size;

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

WorkTime tw;

intime >> tw;

ems->addTime(tw);

while (intime.read((char\*)&tw, sizeof(WorkTime))) {

tw.print\_time();

}

}

intime.close();

cout << "Данные рабочего времени считаны\n" << endl;

}

else {

cout << "Ошибка открытия файла!" << endl;

}

}

При повторном сохранении данных в файл, информация, содержавшаяся в файлах ранее, будет перезаписана новыми данными.

**2.2.6 Описание файлов «Menu.h» и «Menu.cpp»**

В данном заголовочном файле осуществляется подключение стандартных библиотек С++ и описание класса Menu, объекты которого содержат указатели EmployeeManager\* ems, FileManager\* file на объекты классов EmployeeManager и FileManager.

В классе используется перегруженный конструктор с указателями на классы EmployeeManager, FileManager.

Основная функция класса Menu void showMenu() выводит в консоль главное меню и пользователю предлагается выбрать операцию, введя число от 0 до 11. В зависимости от возвращаемого значения переменной choice выполняется тот или иной пункт switch(case) - меню, содержащий методы какого-либо из классов EmployeeManager и FileManager.

Функция содержит бесконечный цикл while (1) {} который выводит меню после каждого завершения работы case.

Case1 позволяет пользователю ввести данные о сотруднике вызывая функцию класса EmployeeManager ems->add\_employee().

Case2 позволяет пользователю ввести данные в табель рабочего времени вызывая функцию класса EmployeeManager ems->add\_time().

С помощью Case3 вызывается функция класса EmployeeManager удаления всех данных ems->deleteAll() и с помощью Case4 можно введя фамилию сотрудника удалить все данные об этом сотруднике:

case 4: {

cout <<"~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;

cout << "\t\tУдалить сотрудника по фамилии " << endl;

cout<<"~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n" << endl;

string surname;

cout << "Введите пожалуйста фамилию сотрудника: ";

cin >> surname;

Employee\* empl = ems->deleteEmployeeBySurname(surname);

cout << "Сотрудник " << surname << " успешно удален!" << endl;

cout << endl;

break;

}

Case5 и Case6 выводить данные о сотрудниках и табель рабочего времени в консоль приложения вызывая функции класса EmployeeManager ems->printEmployee() и ems->printTime().

Case7 выполняет запись данных в файлы «sotrudniki.txt» и «time.txt» вызывая функцию класса FileManager file->SaveToFile(ems) и Case8 выполняет выгрузку данных из файлов вызывая функцию класса FileManager file->LoadFromFile(ems).

Case9 и Case10 выполняют поиск информации сотрудников и записи в табеле рабочего времени по коду сотрудника.

Case11 выполняет выход из программы.

**2.2.7 Описание файла «Source.cpp»**

Файл «Source.cpp» содержит специальную функцию с именем main – это начальная точка любой программы на языке Си или C++.

В данной программе функция main() начинается со строк:

SetConsoleCP(1251);

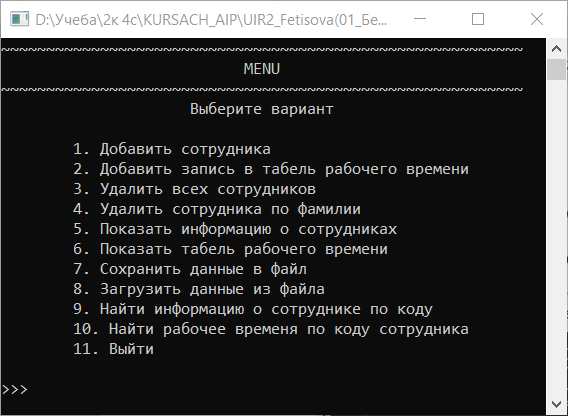
SetConsoleOutputCP(1251);

Эти две функции обеспечивают корректное отображение кириллических символов в консоли и принимают в качестве единственного аргумента номер кодовой страницы кириллицы – 1251. SetConsoleCP() устанавливает нужную кодовую таблицу на поток ввода, а SetConsoleOutputCP() – на поток вывода. Прототипы данных функций содержатся в заголовочном файле «Windows.h».

Функция Main() вызывает функцию showMenu() класса Menu.

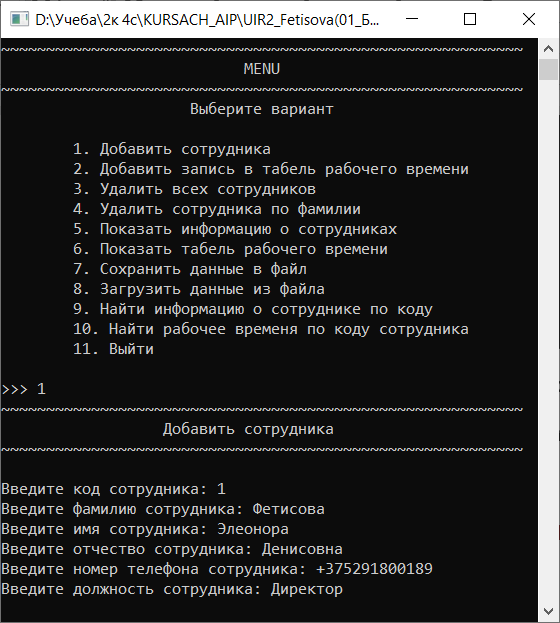
**2.3 Результаты работы программы (скриншоты)**

Первоначальное окно при открытии консоли:



**Рисунок 2.1 – Первоначальное окно консоли**

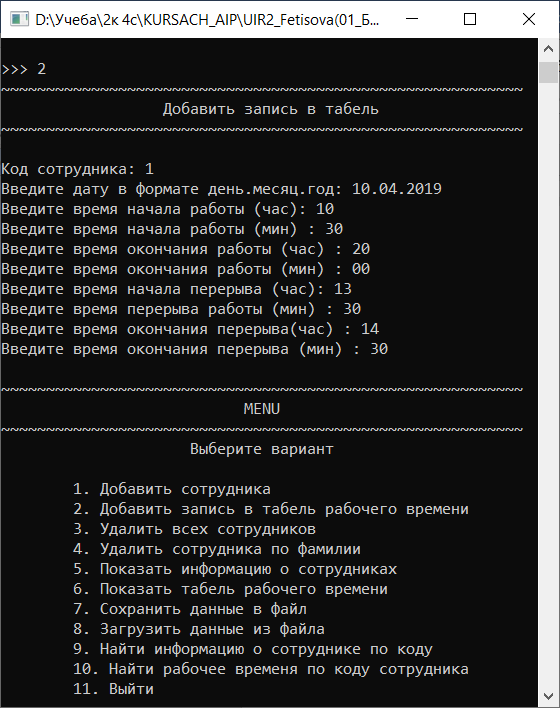
При выборе пункта «1» можно добавить информацию о сотруднике.



**Рисунок 2.2 – При нажатии «1» (case '1')**

При нажатии enter пользователь возвращается в основное меню.

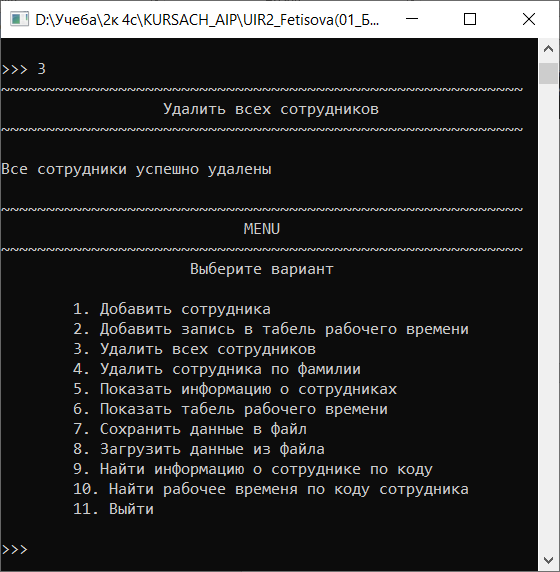
При выборе пункта «2» можно добавить запись в табель рабочего времени.



**Рисунок 2.3 – При нажатии «2» (case '2')**

При нажатии enter пользователь вернется в главное меню.

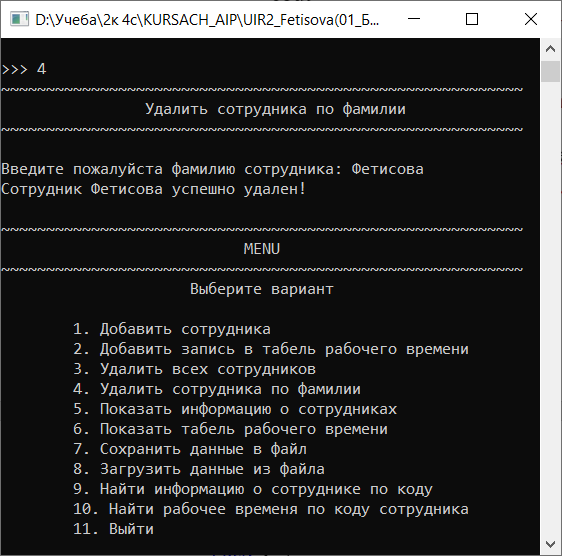
При выборе пункта «3» можно удалить информацию обо всех сотрудниках.



**Рисунок 2.3 – При нажатии «3» (case '3')**

При нажатии enter пользователь вернется в главное меню.

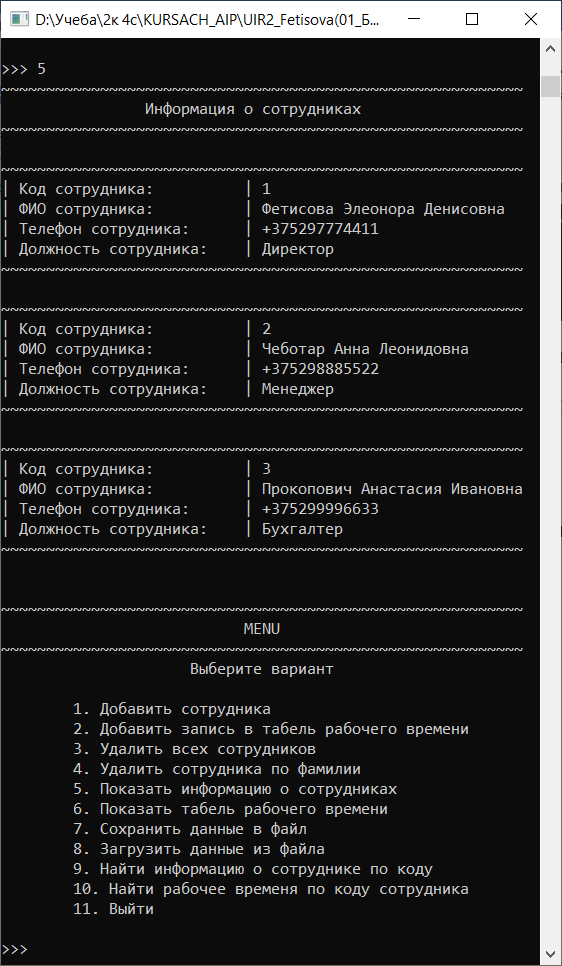
При выборе пункта «4» можно удалить информацию о выбранном сотруднике введя его фамилию.



**Рисунок 2.4 – При нажатии «4» (case '4')**

При нажатии enter пользователь вернется в главное меню.

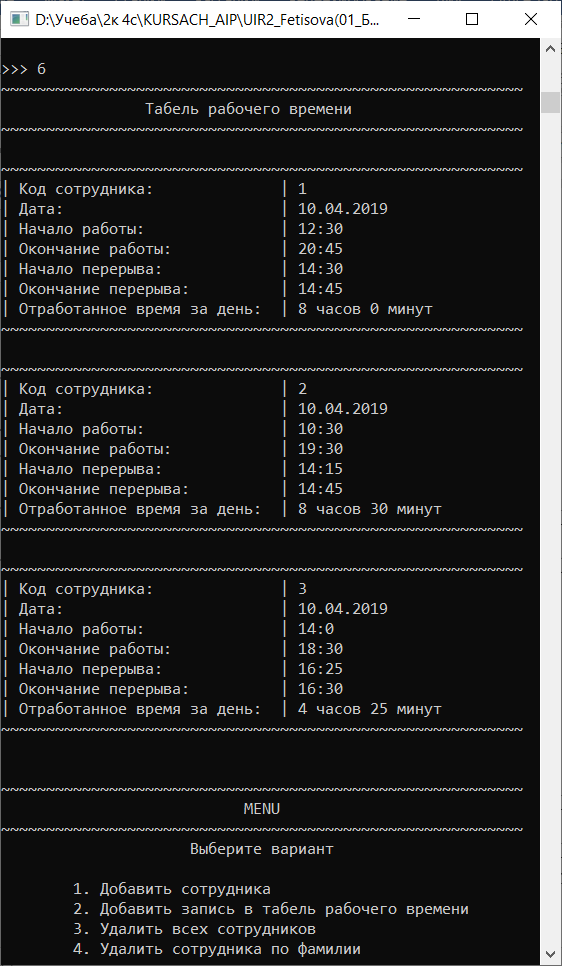
При выборе пункта «5» можно просмотреть информацию обо всех сотрудниках.



**Рисунок 2.5 – При нажатии «5» (case '5')**

При нажатии enter пользователь вернется в главное меню.

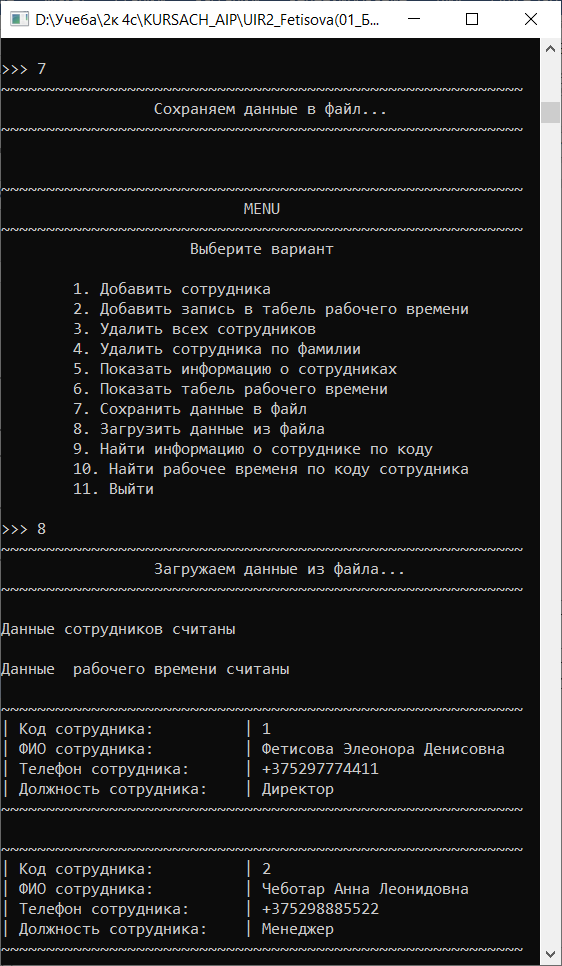
При выборе пункта «6» можно просмотреть табель рабочего времени.



**Рисунок 2.6 – При нажатии «6» (case '6')**

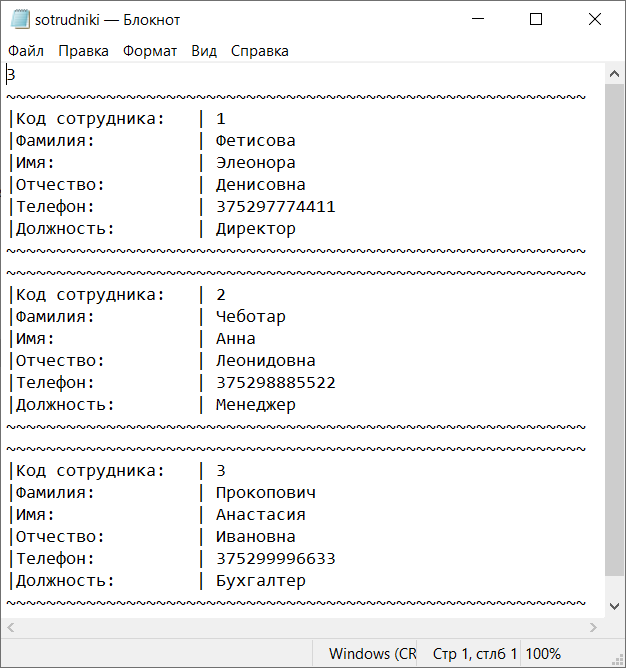
При нажатии enter пользователь вернется в главное меню.

При выборе пункта «7» производится запись данных в файлы «sotrudniki.txt» и «WorkTime.txt».



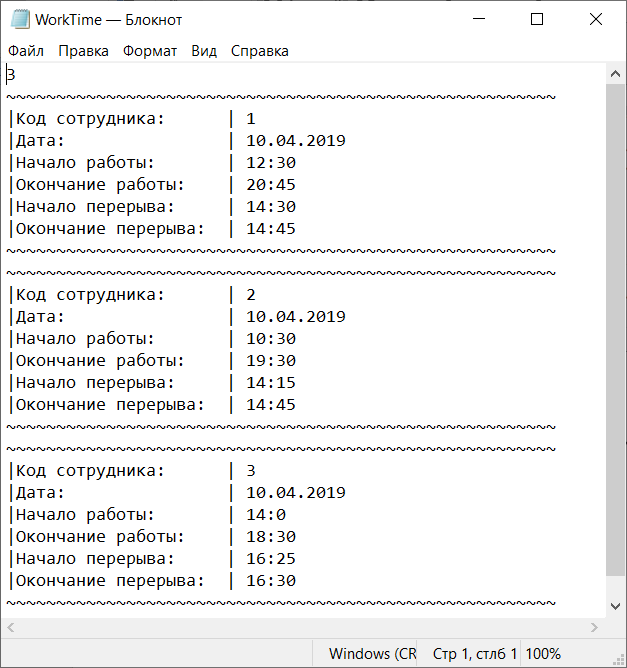
**Рисунок 2.7 – При нажатии «7» (case '7')**

Файл «sotrudniki.txt» выглядит следующим образом:



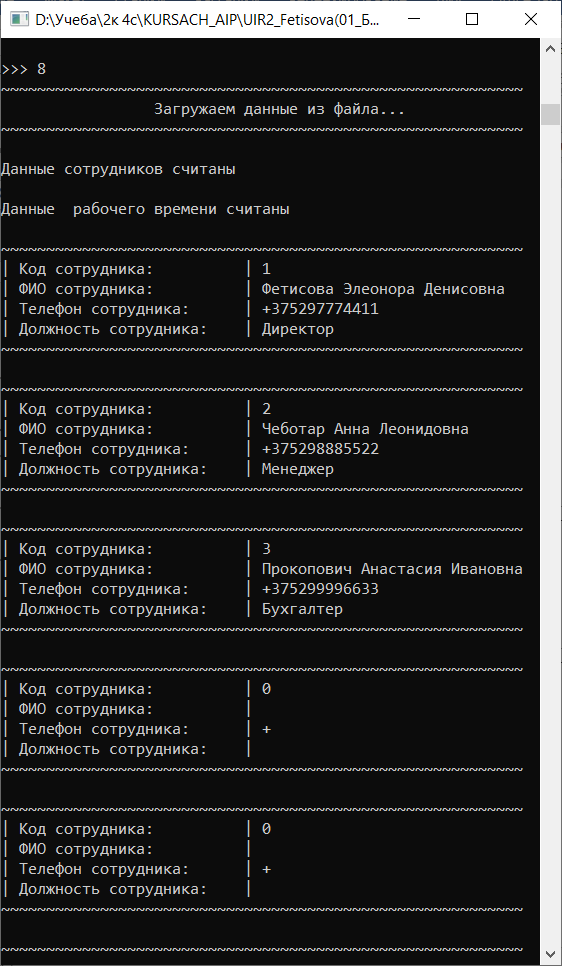
**Рисунок 2.8 – Файл «sotrudniki.txt»**

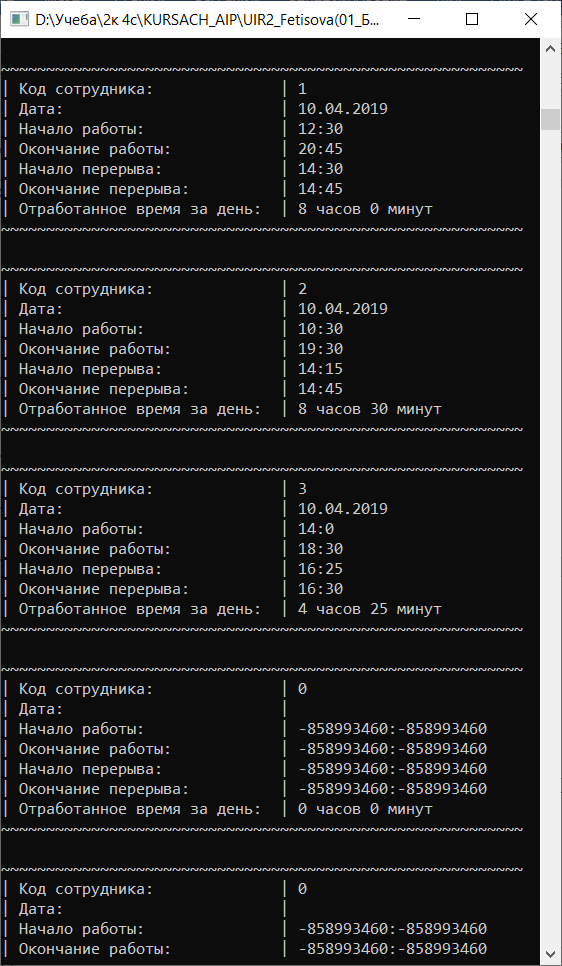
Файл «WorkTime.txt» выглядит следующим образом:



**Рисунок 2.9 – Файл «WorkTime.txt»**

При выборе пункта «8» все данные с файлов считываются.

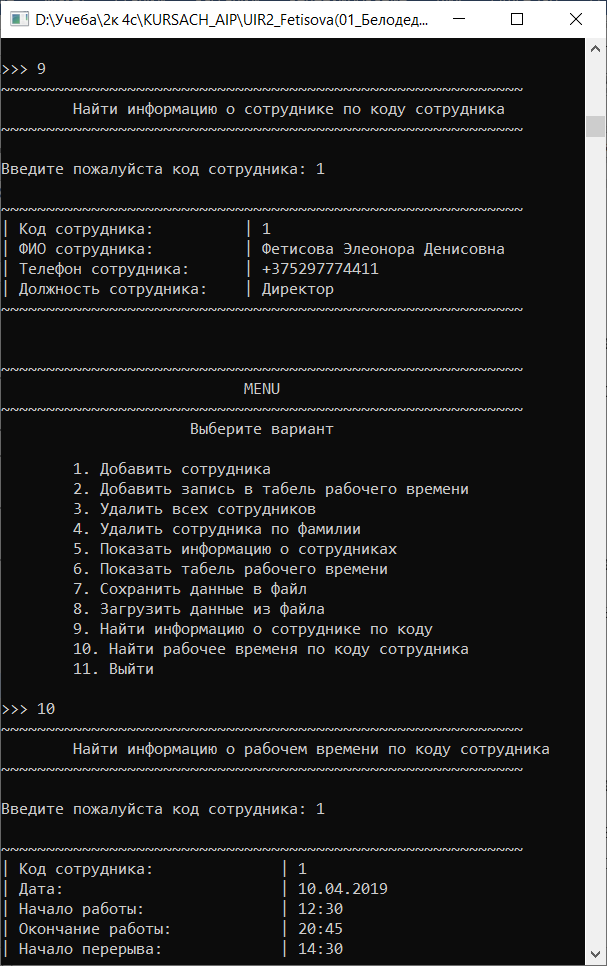




**Рисунок 2.10 – При нажатии «8» (case '8')**

При нажатии enter пользователь вернется в главное меню.

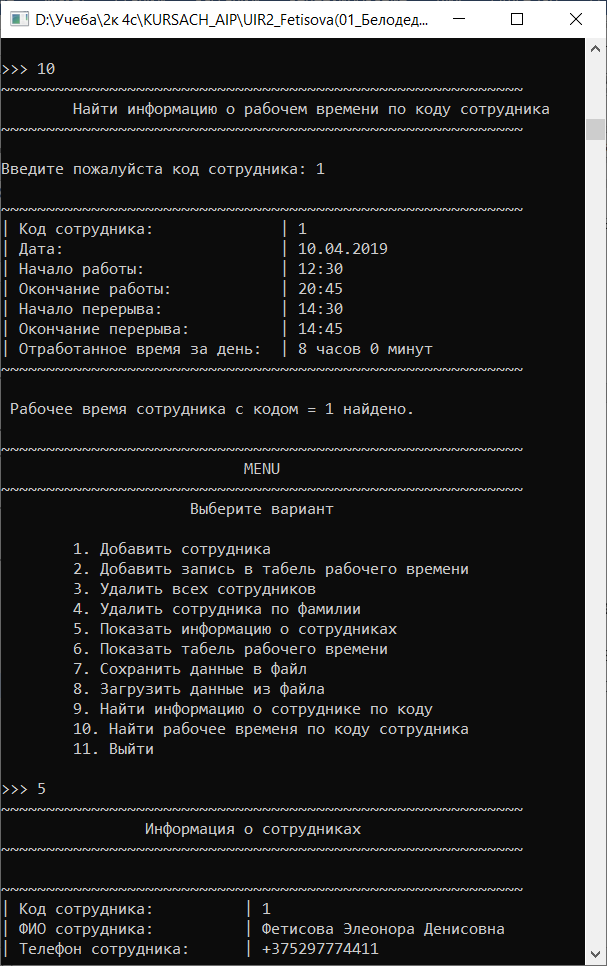
При выборе пункта «9» производится поиск сотрудника при вводе кода сотрудника.



**Рисунок 2.11 – При нажатии «9» (case '9')**

При нажатии enter пользователь вернется в главное меню.

При выборе пункта «10» производится поиск рабочего времени сотрудника по его коду.

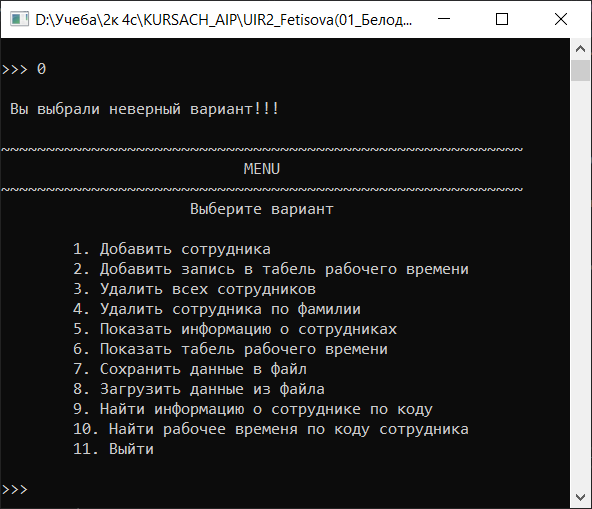


**Рисунок 2.12 – При нажатии «10» (case '10')**

При нажатии enter пользователь вернется в главное меню.

При выборе пункта «11» производится выход и при нажатии enter завершается работа консольного приложения.

При введение неправильного пункта, не входящего в меню, в консоли появится сообщение о неправильном вводе:



**Рисунок 2.13 – При введение неправильного пункта**

**ЗАКЛЮЧЕНИЕ**

В результате выполнения курсовой работы в среде разработки Microsoft Visual Studio 2017 средствами ООП было создано консольное приложение, представляющее собой информационную систему учета рабочего времени сотрудников в организации.

Система позволяет решать такие задачи, как ведение табеля рабочего времени, изменение базы сотрудников, добавление записей о сотрудниках, добавление записей в табель рабочего времени, различные виды поиска данных, сохранение и загрузка данных в файл и из файла.

Работа в системе организована с помощью удобного меню, которое появляется после выполнения каждого действия.

Данная программа подойдет для некрупных компаний с небольшим количеством сотрудников и отсутствием пропускной системы (или любой другой системы автоматического контроля за отработанными часами). Эта программа проста и удобна в использовании и она даст представление о реальной посещаемости и времени, проведенном сотрудником на рабочем месте, что поможет повысить уровень трудовой дисциплины и позволяет значительно сократить потери временных трудозатрат.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Страуструп Б. Язык программирования C++. Специальное издание. Пер. с англ. – М.: Издательство Бином, 2011 г. – 1136 с: ил.
2. Белодед Н.И. Алгоритмизация и программирование: Пособие по курсовому проектированию / Н.И. Белодед - Мн.: Академия управления при Президенте Республики Беларусь, 2005. – 199 с.
3. Каталог API (Microsoft) и справочных материалов [Электронный ресурс]. – Режим доступа: https://msdn.microsoft.com/library. – Дата доступа: 15.05.2017.
4. Шилдт, Г. Самоучитель C++ / Г. Шилдт. – 3-е изд. – СПБ. : БХВ-Петербург, 2002. – 845 с.
5. Основы программирования на языках Си и С++ для начинающих [Электронный ресурс]. – Режим доступа: <http://cppstudio.com/>. – Дата доступа: 15.05.2018.
6. Cplusplus.com [Электронный ресурс]. – Режим доступа: <http://www.cplusplus.com/>.– Дата доступа: 15.05.2018.
7. Онлайн справочник программиста на C и C++ [Электронный ресурс]. – Режим доступа: <http://www.c-cpp.ru/> . – Дата доступа: 07.05.2018.

**ПРИЛОЖЕНИЕ А**

Исходный код программы

// Sotrudniki.h

#pragma once

#include <iostream>

#include <string>

using namespace std;

class Employee {

private:

int employee\_code;

string surname;

string name;

string patronymic;

string number;

string pozition;

public:

Employee();

Employee(int);

Employee(int, string, string, string, string, string);

~Employee();

int getemployee\_code();

string getsurname();

string getname();

string getpatronymic();

string getnumber();

string getpozition();

void print\_employee();

friend ostream& operator << (ostream& out, const Employee& obj);

friend istream& operator >> (istream& in, Employee& obj);

};

// Sotrudniki.cpp

#include "Sotrudniki.h"

Employee::Employee() {}

Employee::Employee(int employee\_code) {

this->employee\_code = employee\_code;

}

Employee::Employee(int employee\_code, string surname, string name, string patronymic, string number, string pozition) {

this->employee\_code = employee\_code;

this->surname = surname;

this->name = name;

this->patronymic = patronymic;

this->number = number;

this->pozition = pozition;

}

Employee::~Employee() {}

int Employee::getemployee\_code() {

return employee\_code;

}

string Employee::getsurname() {

return surname;

}

string Employee::getpatronymic() {

return patronymic;

}

string Employee::getname() {

return name;

}

string Employee::getnumber() {

return number;

}

string Employee::getpozition() {

return pozition;

}

void Employee::print\_employee() {

cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;

cout << "| Код сотрудника: | " << employee\_code << endl;

cout << "| ФИО сотрудника: | " << surname << " " << name << " " << patronymic << endl;

cout << "| Телефон сотрудника: | +" << number << endl;

cout << "| Должность сотрудника: | " << pozition << endl;

cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;

cout << endl;

}

ostream & operator<<(ostream & out, const Employee & obj) {

out << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;

out << "|Код сотрудника: | " << obj.employee\_code << endl;

out << "|Фамилия: | " << obj.surname << endl;

out << "|Имя: | " << obj.name << endl;

out << "|Отчество: | " << obj.patronymic << endl;

out << "|Телефон: | " << obj.number << endl;

out << "|Должность: | " << obj.pozition << endl;

out << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;

return out;

}

istream & operator>>(istream & in, Employee & obj) {

in >> obj.employee\_code;

in >> obj.surname;

in >> obj.name;

in >> obj.patronymic;

in >> obj.number;

in >> obj.pozition;

return in;

}

// Worktime.h

#pragma once

#include <iostream>

#include <string>

using namespace std;

class WorkTime {

private:

int empl\_code;

int StartWorkHour;

int StartWorkMin;

int FinishWorkHour;

int FinishWorkMin;

int StartBreakHour;

int FinishBreakHour;

int StartBreakMin;

int FinishBreakMin;

string date;

public:

WorkTime();

WorkTime(int empl\_code, string date, int StartWorkHour, int StartWorkMin, int FinishWorkHour, int FinishWorkMin, int StartBreakHour, int StartBreakMin, int FinishBreakHour,int FinishBreakMin);

~WorkTime();

int getempl\_code();

int getStartWorkHour();

int getFinishWorkHour();

int getStartWorkMin();

int getFinishWorkMin();

int getStartBreakHour();

int getStartBreakMin();

int getFinishBreakHour();

int getFinishBreakMin();

string getDate();

int GetMinDiff();

int GetHour();

int GetMin();

void print\_time();

friend ostream& operator<< (ostream& outtime, const WorkTime& obj);

friend istream& operator>> (istream& intime, WorkTime& obj);

};

// WorkTime.cpp

#include "WorkTime.h"

WorkTime::WorkTime() {}

WorkTime::WorkTime(int empl\_code, string date, int StartWorkHour, int StartWorkMin, int FinishWorkHour, int FinishWorkMin, int StartBreakHour, int StartBreakMin, int FinishBreakHour, int FinishBreakMin) {

this->empl\_code = empl\_code;

this->date = date;

this->StartWorkHour = StartWorkHour;

this->StartWorkMin = StartWorkMin;

this->FinishWorkHour = FinishWorkHour;

this->FinishWorkMin = FinishWorkMin;

this->StartBreakHour = StartBreakHour;

this->StartBreakMin = StartBreakMin;

this->FinishBreakHour = FinishBreakHour;

this->FinishBreakMin = FinishBreakMin;

}

WorkTime::~WorkTime() { }

int WorkTime::getempl\_code() {

return empl\_code;

}

int WorkTime::getStartWorkHour() {

return StartWorkHour;

}

int WorkTime::getFinishWorkHour() {

return FinishWorkHour;

}

int WorkTime::getStartWorkMin() {

return StartWorkMin;

}

int WorkTime::getFinishWorkMin() {

return FinishWorkMin;

}

int WorkTime::getStartBreakHour() {

return StartBreakHour;

}

int WorkTime::getStartBreakMin() {

return StartBreakMin;

}

int WorkTime::getFinishBreakHour() {

return FinishBreakHour;

}

int WorkTime::getFinishBreakMin() {

return FinishBreakMin;

}

string WorkTime::getDate() {

return date;

}

int WorkTime::GetMinDiff() {

return ((FinishWorkHour - StartWorkHour) \* 60 + (FinishWorkMin - StartWorkMin))-((FinishBreakHour - StartBreakHour) \* 60 + (FinishBreakMin - StartBreakMin));

}

int WorkTime::GetHour() {

return GetMinDiff() / 60;

}

int WorkTime::GetMin() {

return GetMinDiff() % 60;

}

void WorkTime::print\_time() {

cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;

cout << "| Код сотрудника: | " << empl\_code << endl;

cout << "| Дата: | " << date << endl;

cout << "| Начало работы: | " << StartWorkHour << ":" << StartWorkMin << endl;

cout << "| Окончание работы: | " << FinishWorkHour << ":" << FinishWorkMin << endl;

cout << "| Начало перерыва: | " << StartBreakHour << ":" << StartBreakMin << endl;

cout << "| Окончание перерыва: | " << FinishBreakHour << ":" << FinishBreakMin << endl;

cout << "| Отработанное время за день: | " << GetHour() << " часов " << GetMin() << " минут" << endl;

cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;

cout << endl;

}

ostream & operator<<(ostream & outtime, const WorkTime & obj) {

outtime << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;

outtime << "|Код сотрудника: | " << obj.empl\_code << endl;

outtime << "|Дата: | " << obj.date << endl;

outtime << "|Начало работы: | " << obj.StartWorkHour << ": " << obj.StartWorkMin << endl;

outtime << "|Окончание работы: | " << obj.FinishWorkHour << ": " << obj.FinishWorkMin << endl;

outtime << "|Начало перерыва: | " << obj.StartBreakHour << ": " << obj.StartBreakMin<< endl;

outtime << "|Окончание перерыва: | " << obj.FinishBreakHour << ": " << obj.FinishBreakMin << endl;

outtime << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;

return outtime;

}

istream & operator>>(istream & intime, WorkTime & obj) {

intime >> obj.empl\_code;

intime >> obj.date;

intime >> obj.StartWorkHour;

intime >> obj.StartWorkMin;

intime >> obj.FinishWorkHour;

intime >> obj.FinishWorkMin;

intime >> obj.StartBreakHour;

intime >> obj.StartBreakMin;

intime >> obj.FinishBreakHour;

intime >> obj.FinishBreakMin;

return intime;

}

// SotrudnikManager.h

#pragma once

#include <iostream>

#include <string>

#include "Sotrudniki.h"

#include "WorkTime.h"

#define SIZE 10

class EmployeeManager: public Employee, public WorkTime{

private:

int currentEmployeeSize;

int maxEmployeeSize;

Employee\* employee;

int currentTimeWorkSize;

int maxTimeWorkSize;

WorkTime\* time;

public:

EmployeeManager();

EmployeeManager(int employee\_code, string surname, string name, string patronymic, string number, string pozition);

EmployeeManager(int empl\_code, string date, int StartWorkHour, int StartWorkMin, int FinishWorkHour, int FinishWorkMin, int StartBreakHour, int StartBreakMin, int FinishBreakHour, int FinishBreakMin);

~EmployeeManager();

void add\_employee();

void add\_time();

Employee\* findEmployeeByCode(int);

WorkTime\* findWorkTimeByEmployeeCode(int);

void printEmployee();

EmployeeManager\* printTime();

void deleteAll();

Employee\* deleteEmployeeBySurname(string);

Employee\* getEmployee();

WorkTime\* getTime();

void addEmployee(Employee empl);

void addTime(WorkTime tw);

int getEmployeeCurrentSize();

int getTimeCurrentSize();

};

// SotrudnikManager.cpp

#include "SotrudnikManager.h"

EmployeeManager::EmployeeManager() {

currentEmployeeSize = 0;

maxEmployeeSize = SIZE;

employee = new Employee[maxEmployeeSize];

currentTimeWorkSize = 0;

maxTimeWorkSize = SIZE;

time = new WorkTime[maxTimeWorkSize];

}

EmployeeManager::EmployeeManager(int employee\_code, string surname, string name, string patronymic, string number, string pozition) : Employee(employee\_code, surname, name, patronymic, number, pozition) {}

EmployeeManager::EmployeeManager(int empl\_code, string date, int StartWorkHour, int StartWorkMin, int FinishWorkHour, int FinishWorkMin, int StartBreakHour, int StartBreakMin, int FinishBreakHour, int FinishBreakMin) : WorkTime(empl\_code, date, StartWorkHour, StartWorkMin, FinishWorkHour, FinishWorkMin, StartBreakHour, StartBreakMin, FinishBreakHour, FinishBreakMin) {}

EmployeeManager::~EmployeeManager() {

delete[] employee;

delete[] time;

}

void EmployeeManager::add\_employee() {

int employee\_code;

string surname;

string name;

string patronymic;

string number;

string pozition;

cout << "Введите код сотрудника: ";

cin >> employee\_code;

cout << "Введите фамилию сотрудника: ";

cin >> surname;

cout << "Введите имя сотрудника: ";

cin >> name;

cout << "Введите отчество сотрудника: ";

cin >> patronymic;

cout << "Введите номер телефона сотрудника: +";

cin >> number;

cout << "Введите должность сотрудника: ";

cin >> pozition;

Employee sup(employee\_code, surname, name, patronymic, number, pozition);

addEmployee(sup);

}

void EmployeeManager::add\_time() {

int StartWorkHour;

int StartWorkMin;

int FinishWorkHour;

int FinishWorkMin;

int StartBreakHour;

int StartBreakMin;

int FinishBreakHour;

int FinishBreakMin;

int empl\_code;

string date;

cout << "Код сотрудника: ";

cin >> empl\_code;

cout << "Введите дату в формате день.месяц.год: ";

cin >> date;

cout << "Введите время начала работы (час): ";

cin >> StartWorkHour;

cout << "Введите время начала работы (мин) : ";

cin >> StartWorkMin;

cout << "Введите время окончания работы (час) : ";

cin >> FinishWorkHour;

cout << "Введите время окончания работы (мин) : ";

cin >> FinishWorkMin;

cout << "Введите время начала перерыва (час): ";

cin >> StartBreakHour;

cout << "Введите время перерыва работы (мин) : ";

cin >> StartBreakMin;

cout << "Введите время окончания перерыва(час) : ";

cin >> FinishBreakHour;

cout << "Введите время окончания перерыва (мин) : ";

cin >> FinishBreakMin;

WorkTime tw(empl\_code, date, StartWorkHour, StartWorkMin, FinishWorkHour, FinishWorkMin, StartBreakHour, StartBreakMin, FinishBreakHour, FinishBreakMin);

addTime(tw);

}

void EmployeeManager::printEmployee() {

for (int i = 0; i < currentEmployeeSize; i++) {

employee[i].print\_employee();

}

}

EmployeeManager\* EmployeeManager::printTime() {

for (int i = 0; i < currentTimeWorkSize; i++) {

time[i].print\_time();

}

return nullptr;

}

// поиск сотрудника по коду

Employee\* EmployeeManager::findEmployeeByCode(int employee\_code) {

for (int i = 0; i < currentEmployeeSize; i++) {

if (employee[i].getemployee\_code() == employee\_code) {

return &employee[i];

}

}

return nullptr;

}

// найти рабочее время сотрудника по коду

WorkTime\* EmployeeManager::findWorkTimeByEmployeeCode(int employee\_code) {

for (int i = 0; i < currentTimeWorkSize; i++) {

if (time[i].getempl\_code() == employee\_code) {

return &time[i];

}

}

return nullptr;

}

// удаление всех сотрудников

void EmployeeManager::deleteAll() {

currentEmployeeSize = 0;

cout << "Все сотрудники успешно удалены" << endl;

}

// удаление сотрудника по фамилии

Employee\* EmployeeManager::deleteEmployeeBySurname(string surname) {

for (int i = 0; i < currentEmployeeSize; i++) {

if (employee[i].getsurname() == surname) {

for (int j = i; j < currentEmployeeSize - 1; j++) {

employee[j] = employee[j + 1];

}

currentEmployeeSize--;

}

}

return nullptr;

}

Employee\* EmployeeManager::getEmployee() {

return &employee[0];

}

WorkTime\* EmployeeManager::getTime() {

return &time[0];

}

void EmployeeManager::addEmployee(Employee empl) {

if (currentEmployeeSize == maxEmployeeSize) {

Employee\* temp = employee;

int newSize = 2 \* maxEmployeeSize + 1;

employee = new Employee[newSize];

for (int i = 0; i < maxEmployeeSize; i++) {

\*(employee + i) = \*(temp + i);

}

delete[] temp;

maxEmployeeSize = newSize;

}

employee[currentEmployeeSize++] = empl;

}

void EmployeeManager::addTime(WorkTime tw) {

if (currentTimeWorkSize == maxTimeWorkSize) {

WorkTime\* temp = time;

int newSize = 2 \* maxTimeWorkSize + 1;

time = new WorkTime[newSize];

for (int i = 0; i < maxTimeWorkSize; i++) {

\*(time + i) = \*(temp + i);

}

delete[] temp;

maxTimeWorkSize = newSize;

}

time[currentTimeWorkSize++] = tw;

}

int EmployeeManager::getEmployeeCurrentSize() {

return currentEmployeeSize;

}

int EmployeeManager::getTimeCurrentSize() {

return currentTimeWorkSize;

}

//FileManager.h

#pragma once

#include <stdio.h>

#include <iostream>

#include <string>

#include "SotrudnikManager.h"

#include <fstream>

using namespace std;

class FileManager {

public:

void LoadFromFile(EmployeeManager\* ems);

void SaveToFile(EmployeeManager\* ems);

};

// FileManager.cpp

#include "FileManager.h"

// сохранить все данные о сотрудниках в файл

void FileManager::SaveToFile(EmployeeManager\* ems){

ofstream out("sotrudniki.txt");

if (out.is\_open()) {

int size = ems->getEmployeeCurrentSize();

out << size << endl;

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

out << ems->getEmployee()[i];

}

out.close();

}

else {

cout << "Ошибка открытия файла!" << endl;

}

ofstream outtime("WorkTime.txt");

if (outtime.is\_open()) {

int size = ems->getTimeCurrentSize();

outtime << size << endl;

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

outtime << ems->getTime()[i];

}

outtime.close();

}

else {

cout << "Ошибка открытия файла!" << endl;

}

}

// загрузка всех данных о сотрудниках из файла

void FileManager::LoadFromFile(EmployeeManager\* ems){

ifstream in("sotrudniki.txt");

if (in.is\_open()) {

int size;

in >> size;

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

Employee empl;

in >> empl;

ems->addEmployee(empl);

while (in.read((char\*)&empl, sizeof(Employee))) {

empl.print\_employee();

}

}

in.close();

cout << "Данные сотрудников считаны\n" << endl;

}

else {

cout << "Ошибка открытия файла!" << endl;

}

ifstream intime("WorkTime.txt");

if (intime.is\_open()) {

int size;

intime >> size;

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

WorkTime tw;

intime >> tw;

ems->addTime(tw);

while (intime.read((char\*)&tw, sizeof(WorkTime))) {

tw.print\_time();

}

}

intime.close();

cout << "Данные рабочего времени считаны\n" << endl;

}

else {

cout << "Ошибка открытия файла!" << endl;

}

}

// Menu.h

#pragma once

#include <iostream>

#include <string>

#include "SotrudnikManager.h"

#include "FileManager.h"

class Menu {

private:

EmployeeManager\* ems;

FileManager\* file;

public:

Menu();

Menu(EmployeeManager\* ems, FileManager\* file);

void showMenu();

};

// Menu.cpp

#include "Menu.h"

Menu::Menu() {}

Menu::Menu(EmployeeManager\* ems, FileManager\* file) {

this->ems = ems;

this->file = file;

}

void Menu::showMenu() {

while (1) {

int choice;

cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;

cout << "\t\t\t MENU " << endl;

cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;

cout << "\t\t Выберите вариант\n" << endl;

cout << "\t1. Добавить сотрудника\n"

<< "\t2. Добавить запись в табель рабочего времени\n"

<< "\t3. Удалить всех сотрудников\n"

<< "\t4. Удалить сотрудника по фамилии\n"

<< "\t5. Показать информацию о сотрудниках\n"

<< "\t6. Показать табель рабочего времени\n"

<< "\t7. Сохранить данные в файл\n"

<< "\t8. Загрузить данные из файла\n"

<< "\t9. Найти информацию о сотруднике по коду\n"

<< "\t10. Найти рабочее временя по коду сотрудника\n"

<< "\t11. Выйти\n" << endl;

cout << ">>> ";

cin >> choice;

switch (choice) {

case 1: {

cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;

cout << "\t\t Добавить сотрудника " << endl;

cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n" << endl;

ems->add\_employee();

cout << endl;

break;

}

case 2: {

cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;

cout << "\t\t Добавить запись в табель " << endl;

cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n" << endl;

ems->add\_time();

cout << endl;

break;

}

case 3: {

cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;

cout << "\t\t Удалить всех сотрудников " << endl;

cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n" << endl;

ems->deleteAll();

cout << endl;

break;

}

case 4: {

cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;

cout << "\t\tУдалить сотрудника по фамилии " << endl;

cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n" << endl;

string surname;

cout << "Введите пожалуйста фамилию сотрудника: ";

cin >> surname;

Employee\* empl = ems->deleteEmployeeBySurname(surname);

cout << "Сотрудник " << surname << " успешно удален!" << endl;

cout << endl;

break;

}

case 5: {

cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;

cout << " Информация о сотрудниках " << endl;

cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n" << endl;

ems->printEmployee();

cout << endl;

break;

}

case 6: {

cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;

cout << " Табель рабочего времени " << endl;

cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n" << endl;

ems->printTime();

cout << endl;

break;

}

case 7: {

cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;

cout << "\t\t Сохраняем данные в файл..." << endl;

cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n" << endl;

file->SaveToFile(ems);

cout << endl;

break;

}

case 8: {

cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;

cout << "\t\t Загружаем данные из файла..." << endl;

cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n" << endl;

file->LoadFromFile(ems);

ems->printEmployee();

ems->printTime();

cout << endl;

break;

}

case 9: {

cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;

cout << "\tНайти информацию о сотруднике по коду сотрудника " << endl;

cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n" << endl;

int code;

cout << "Введите пожалуйста код сотрудника: ";

cin >> code;

cout << endl;

Employee\* empl = ems->findEmployeeByCode(code);

if (empl != nullptr) {

empl->print\_employee();

}

else {

cout << " Cотрудник с кодом = " << code << " не найден." << endl;

}

cout << endl;

break;

}

case 10: {

cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;

cout << "\tНайти информацию о рабочем времени по коду сотрудника " << endl;

cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n" << endl;

int code;

cout << "Введите пожалуйста код сотрудника: ";

cin >> code;

cout << endl;

WorkTime\* time = ems->findWorkTimeByEmployeeCode(code);

if (time != nullptr) {

cout << "Рабочее время сотрудника с кодом = " << code << " найдено." << endl;

}

else {

cout << " Рабочее время сотрудника с кодом = " << code << " не найдено." << endl;

}

cout << endl;

break;

}

case 11: {

return;

}

default: {

cerr << " Вы выбрали неверный вариант!!!" << endl << endl;

break;

}

}

}

}

// Source.cpp

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <Windows.h>

#include <string>

#include "Menu.h"

using namespace std;

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

EmployeeManager ems;

FileManager file;

Menu menu(&ems, &file);

menu.showMenu();

cout << endl;

system("pause");

return 0;

}

**ПРИЛОЖЕНИЕ Б**

Алгоритм (блок-схема) программы















