# АННОТАЦИЯ

В пояснительной записке описывается программа для операционной системы Windows, предназначенная для обработки данных о служащих завода.

Документ содержит описание назначения, технических требований, описание работы и реализации программы. Так же в документе предоставлено детальное пояснение структуры программы, её алгоритмов и особенностей работы на ЭВМ. В пояснительную записку включено руководство к эксплуатации программы и графическое представление программы в виде структурных схем алгоритмов.

# СОДЕРЖАНИЕ

[АННОТАЦИЯ 3](#_Toc500365182)

[СОДЕРЖАНИЕ 4](#_Toc500365183)

[ВВЕДЕНИЕ 5](#_Toc500365184)

[1. НАЗНАЧЕНИЕ И ОБЛАСТЬ ПРИМЕНЕНИЯ ПРОГРАММЫ 6](#_Toc500365185)

[2. ТЕХНИЧЕСКИЕ ХАРАКТЕРИСТИКИ ПРОГРАММЫ 7](#_Toc500365186)

[2.1 Постановка задачи на разработку программы. 7](#_Toc500365187)

[2.2 Применяемые математические методы 8](#_Toc500365188)

[2.3 Описание и обоснование выбора метода организации входных, выходных и промежуточных данных. 8](#_Toc500365189)

[2.4 Разработка модульной структуры программы. 10](#_Toc500365190)

[2.5 Описание функций используемых в программе. 10](#_Toc500365191)

[2.6 Описание алгоритмов функционирования программы. 15](#_Toc500365192)

[2.7 Обоснование состава технических и программных средств. 33](#_Toc500365193)

[3. ВЫПОЛНЕНИЕ ПРОГРАММЫ 35](#_Toc500365194)

[3.1 Условия выполнения программы 35](#_Toc500365195)

[3.2 Загрузка и запуск программы. 35](#_Toc500365196)

[3.3 Проверка работоспособности программы. 42](#_Toc500365197)

[ВЫВОД 45](#_Toc500365198)

[ПЕРЕЧЕНЬ ССЫЛОК 46](#_Toc500365199)

[Приложение А 47](#_Toc500365200)

# ВВЕДЕНИЕ

В рамках курсового проектирования ведется разработка программы обработки данных служащих завода и расчёта их средней заработной платы по участкам. Разработка программы ведется на основании технического задания, выданного 02 сентября 2017 года кафедрой информационных систем Севастопольского государственного университета.

Программа была написана на языках C/C++ с использованием принципов структурного программирования. Написание исходного кода производилось в интегрированной среде программирования CLion компании Jet Brains.

Данная программа предусматривает возможность манипуляции с полями данных сотрудников завода. Поддерживает ввод данных с клавиатуры и вывод их на экран, а так же ввод/вывод данных служащих в текстовый или бинарный файл.

# НАЗНАЧЕНИЕ И ОБЛАСТЬ ПРИМЕНЕНИЯ ПРОГРАММЫ

Назначение программы заключается в обработке данных служащих завода, в частности: фамилии и инициалов, года рождения, пола, профессии, номера цеха, номера участка и заработной платы. Программа позволяет вводить, а так же редактировать данные, упорядочивать их по предусмотренным критериям, а так же выполнять полный и выборочный вывод данных на экран или файл, при этом размер файла будет незначительным. Сохраненный на жестком диске файл может быть загружен в программу и использоваться в качестве исходных данных. Для корректной работы программы, необходим объём оперативной памяти пропорциональный количеству записей в базе.

Разработанная программа может быть запущена в любой операционной системе семейства Windows, начиная с Windows XP, и использована на любых производственных предприятиях и заводах.

# ТЕХНИЧЕСКИЕ ХАРАКТЕРИСТИКИ ПРОГРАММЫ

1. Постановка задачи на разработку программы.

* Разрабатываемая программа должна использовать меню-ориентированный интерфейс, обеспечивающий выполнение минимального состава действий:
* Начальное создание таблицы.
* Просмотр таблицы. При этом необходимо предусмотреть возможность скроллинга.
* Добавление новой записи в таблицу.
* Удаление записи. Удаляемый элемент выбирается по одному из полей таблицы (ключевому).
* Корректировка записи в таблице. Корректируемую запись выбирают по одному из полей таблицы.
* Сортировка таблицы. Сортировка производится по одному из полей таблицы (ключевому).
* Поиск записи в таблице по ключевому полю.
* Поиск группы записей в таблице по ключевому полю. Вывод данных группы в файл.
* Сохранение таблицы в текстовом и бинарном файле. Имя файла должен вводить пользователь. Сохранение таблицы в текстовом файле обеспечит при необходимости возможность её печати.
* При необходимости создания новой таблицы исходные данные могут считываться из текстового или бинарного файла. Имя файла должен задавать пользователь.
* Обработка таблицы и просмотр результатов обработки. Результат обработки необходимо вывести на экран и в текстовый файл. Имя файла вводит пользователь.
* Выход ― завершение работы программы.

Структура записей входного массива имеет следующий вид: табельный номер, Ф.И.О. (30 символов), год рождения, пол (булевская переменная), профессия (10 символов), стаж работы, разряд рабочего, номер цеха, номер участка, сумма заработной платы. Вид выходной таблицы представлен в таб. 2.1.

Таблица 2.1 – Вид выходной таблицы согласно варианту

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № участка | Средняя зарплата | Количество рабочих по разрядам | | | Всего |
| 1 | 2 | 3 |
|  |  |  |  |  |  |

1. Применяемые математические методы

Для упорядочивания элементов в таблице, используется алгоритм основанный на методе сортировке прямым выбором.

Для подсчёта средней заработной платы на участках завода используется математический метод расчёта среднего арифметического чисел.

1. Описание и обоснование выбора метода организации входных, выходных и промежуточных данных.

Для хранения информации в программе была выбрана динамическая структура данных – двунаправленная очередь, каждый элемент которой состоит из информационного поля – структурной переменной, хранящей данные о сотруднике, указателя на следующий элемент и указателя на следующий элемент.

struct list {

tableData inf;

struct list \*next, \*pred;

}

struct tableData {

unsigned int personalNumber;

unsigned short exp, rank, factoryNumber, deportmentNumber;

char fio[FIO\_LENGTH], prof[PROF\_LENGTH];

unsigned short birth\_year;

bool sex;

float salary;

}

Выбор данной динамической структуры был сделан для удобства навигации между элементами в функциях вывода данных, использующих интерфейс, а так же для снижения затрат памяти, так как при использовании таких структур нет необходимости в использовании рекурсивный функций, к примеру как при использовании бинарных деревьев.

Модификатор unsigned был использован для полей содержащих целочисленные значения, так как в данных полях нет необходимости использовать отрицательные значения.

В полях birth\_day, personalNumber, exp, rank, factoryNumber, deportmentNumber использован тип short, так как максимальное значение хранимое в переменной данного типа во много раз превышает допустимые значения полей, используемых в программе.

Поле fio представлено массивом символов, который содержит 21 ячейку. Последняя ячейка содержит символ ‘\0’ – обозначающий символ конца строки. Аналогичным образом представлено поле prof, но его размер равен 11 ячейкам.

Поле sex представлено булевой переменной, и содержит пол работника. Значения поля: 0 – мужской, 1 – женский.

Для разработки и написания программы была использована интегрированная среда разработки CLion, поддерживаемая компанией Jet Brains. Данная среда предоставляет широкий набор инструментов: подсветку синтаксиса, авто форматирование кода, подсказки при наборе, а так же встроенный инструмент для работы с системами контроля версий.

1. Разработка модульной структуры программы.

Разработка данной программы производилась в интегрированной среде разработки Clion. Она представлена как проект, содержащий файлы:

* main.cpp
* CP\_main\_menu.cpp
* CP\_types.h
* CP\_consmanip.cpp
* CP\_listmanip.cpp
* CP\_interface.cpp
* CP\_filemanip.cpp
* CP\_checks.cpp
* CP\_sort.cpp

Кодирование программы начиналось с файла «CP\_types.h», в котором описаны основные структуры, константы, а так же заголовки функций используемых в проекте. Так же в данном файле подключены необходимые стандартные библиотеки языка C.

Объявления функций разбито на группы, в зависимости от того над чем производятся манипуляции, например: файлы, динамические структуры данных, вывод данных на экран, прорисовка интерфейса и т.п. Следуя этому же принципу, проект разбит на файлы.

1. Описание функций используемых в программе.

void gotoxy(short x, short y):

* устанавливает курсор, относительно левого верхнего угла консоли, в положение переданное в параметрах

void SetColor(int text, int background):

* устанавливает цвет текста и фона в консоли;
* для возврата цвет по умолчанию необходимо вызвать данную функцию с параметрами 7,0

tableData newRecord():

* функция для заполнения данных структуры элемента
* возвращает заполненную структуру
* если данные поля не введены присваивается значению:
* текстовое - ‘\0’:
* числовое - 0;

int organizeList(list \*&top, list \*&end, tableData personalData):

* функция для организации очереди
* позволяет добавить только первый элемент
* возвращает:
* 0 - в случае добавления
* 1 - в случае не пустого списка

int addPerson(list \*&end, tableData personalData):

* функция добавления элемента в очередь
* добавляет элементы только, если список не пуст
* возвращает:
* 0 - в случае добавления
* 1 - в случае пустого списка

tableData editData(tableData mainData):

* функция редактирование записи
* при нажатии клавиши:
* ~ - возвращает изменённую структуру
* esc - структуру без изменений

int deleteList(list \*&top):

* функция удаления списка
* возвращает:
* 0 - в случае успешного удаления
* 1 - в случае пустого списка

int deletePersonalData(list \*&listHead, list \*&listEnd, list \*current):

* функция удаления одного элемента списка

void viewList(list \*&listHead, list \*&listEnd, unsigned mode):

* функция просмотра очереди с возможностью навигации(по записям, а так же по страницам)
* mode:
* 0 - функция для удаления элементов
* 1 - для сохранения выведенного списка в файл

int saveFile(list \*top, char \*fileName, bool mode):

* функция сохранения очереди в файл
* mode:
* 0 - текстовый файл или если файл имеет расширение ".txt"
* 1 - бинарный файл или если файл имеет расширение ".bin"

int loadFile(list \*&top, list \*&end, char \*fileName):

* функция загрузки очереди из файла

void saveFileInterface():

* интерфейс для функции сохранения

void loadFileInterface():

* функция для вызова функции загрузки файла

void outData(list \*temp):

* функция вывода данных в консоль(в одну строку)

int menu(const char \*\*menuItems, const int itemsCount, int currentItem):

* функция вывода и навигации основного меню
* возвращает номер пункта меню в переданном массиве

int menuInterface(const char \*\*menuItems, const int itemsCount):

* функция вывода и навигации дополнительного меню(сохранения в файл или подтверждения операций)
* возвращает номер пункта меню в переданном массиве

void drawHelpMenu(unsigned mode):

* функция печати дополнительного меню
* mode:
* 0 - выводит пункт "del - dlt";
* 1 - выводит пункт " ~ - save";

void drawTableHead(unsigned X, unsigned Y):

* функция вывода шапки таблицы

void cleanPlace():

* функция очистки консоли ниже status bar

void cleanStatusBar():

* функция очистки status bar

void emptyMessage():

* функция выводит сообщение в случае пустого списка

unsigned int checkNumeral(short X, short Y, long int num, int maxDigitCount):

* функция проверки числа на ввод, с возможностью выбора координат
* при передаче x < 0 и y < 0 - координаты не учитываются
* при нажатии esc - возвращается начальное значение переданное функции или 0

char \*strToFormat(char \*str, const int length):

* приведение символов переданного массива к верхнему регистру(учитывая русские символы)

int rewriteString(unsigned X, unsigned Y, const int length, char \*str):

* функция проверки символов на ввод, с возможностью выбора координат
* при передаче x<0 и y <0 - координаты не учитываются
* при нажатии esc - возвращается 0 и начальное значение строки не изменяется

unsigned checkPersonalNumber(int num, list \*top):

* функция проверки совпадения индивидуального номера при вводе данных
* возвращает 1 - при совпадении номеров, 0 - в противном случае

int sort(list \*&head, list \*&end, short int mode):

* функция сортировки списка по критериям
* mode:
* 0 - по фамилии
* 1 - по индивидуальному номеру
* 2 - по профессии и фамилии
* 3 - по номеру цеха(factory) и фамилии
* 4 - по номеру участка(deportment) и фамилии
* 5 - по заработной плате(salary) и фамилии
* 6 - по полу(sex) и фамилии
* 7 - по профессии и разряду

int searchData(list \*&head, short int mode):

* функция для поиска данных по ключевому полю
* возвращает 1 - в случае пустого списка

void registerOfWorkers(list \*head):

* функция подсчёта средней зарплаты и количества работников на участке
* сохраняет данные в файл workers\_register.txt
* выводит содержимое файла на экран

1. Описание алгоритмов функционирования программы.

Далее представлены структурные схемы алгоритмов функций используемых в программе и их поблочное описание.



Рисунок 2.1 – Структурная схема алгоритма функции organizeList(list \*&top, list \*&end, tableData personalData)

Поблочное описание:

Блок 1: Проверка списка на наличие элементов

Блок 2: Выделение памяти под структуру данных

Блок 3: Копирование переданного значения в новый элемент

Блок 4: Связывание указателей next, pred

Блок 5: Возвращаемое значение, при успешном добавлении элемента

Блок 6: Возвращаемое значение, при существовании списка



Рисунок 2.2 – Структурная схема алгоритма функции

deleteList(list \*&top)

Поблочное описание:

Блок 1: Проверка списка на наличие элементов

Блок 2: Проверка наличия элементов в списке

Блок 3: Удаление элемента списка

Блок 4: Связывание указателей next, pred



Рисунок 2.3 – Структурная схема алгоритма функции

deletePersonalData(list \*&listHead, list \*&listEnd, list \*current)

Поблочное описание:

Блок 1: Проверка является ли удаляемый элемент единственным в списке

Блок 2: Удаления указателей начала и конца списка

Блок 3: Освобождение памяти от удаляемого элемента

Блок 4: Проверка является ли удаляемый элемент первым в списке

Блок 5: Переназначение указателя на начало списка и его указателя pred

Блок 6: Освобождение памяти от удаляемого элемента

Блок 7: Проверка является ли удаляемый последним в списке

Блок 8: Переназначение указателя на конец списка и его указателя next

Блок 9: Освобождение памяти от удаляемого элемента

Блок 10: Связывание указателей предыдущего и следующего элементов

Блок 11: Освобождение памяти от удаляемого элемента



Рисунок 2.4 – Структурная схема алгоритма функции

addPerson(list \*&end, tableData personalData)

Поблочное описание:

Блок 1: Проверка списка на существование

Блок 2: Выделение памяти под структуру данных

Блок 3: Копирование переданного значения в новый элемент

Блок 4: Связывание указателей next, pred, переопределение указателя на последний элемент

Блок 5: Возвращаемое значение, при успешном добавлении элемента

Блок 6: Возвращаемое значение, при отсутствии элементов в списке



Рисунок 2.5 – Структурная схема алгоритма функции

newRecord()

Поблочное описание:

Блок 1: Считывание нажатия клавиши для продолжения ввода

Блок 2: Проверка кода клавиши, если нажата клавиша ESC, осуществляется выход из подпрограммы

Блок 3: Считывание ФИО при помощи функции rewriteString(unsigned X, unsigned Y, const int length, char \*str) – которая позволяет вводить только буквы англ. и русского алфавита и ограничивающая количество введённых букв.

Блок 4: Преобразование введённой строки к верхнему регистру при помощи функции strToFormat(char \*str, const int length)

Блок 5: Ввод табельного номера с ограничением количества разрядов в числе до 6

Блок 6: Вызов функции для проверки табельного номера checkPersonalNumber(int num, list \*top)

Блок 7: Проверка результата функции checkPersonalNumber(int num, list \*top)

Блок 8: Проверка корректности введённого года рождения

Блок 9: Ввод года рождения с ограничением количества разрядов в числе до 4

Блок 10: Проверка корректности введённой информации о половой принадлежности

Блок 11: Ввод информации о половой принадлежности с ограничением на ввод, принимает значение только 0 или 1

Блок 12: Считывание профессии при помощи функции rewriteString(unsigned X, unsigned Y, const int length, char \*str)

Блок 13: Ввод информации о стаже работы с ограничением количества разрядов в числе до 2

Блок 14: Проверка корректности введённого разряда

Блок 15: Ввод информации о разряде работы с ограничением количества разрядов в числе до 1

Блок 16: Ввод номера цеха с ограничением количества разрядов в числе до 2

Блок 17: Проверка корректности введённого номера участка

Блок 18: Ввод номера участка с ограничением количества разрядов в числе до 2

Блок 19: Ввод размера зарплаты с ограничением количества разрядов в числе до 7

Блок 20: Возвращаемое значение функции соответствует заполненной структуре



Рисунок 2.6 – Структурная схема алгоритма функции

registerOfWorkers(list \*head)

Поблочное описание:

Блок 1: Определение новой структуры данных для хранения информации о количестве рабочих пред-я на участках поразрядно и их заработной плате

Блок 2: Считывание данных из списка

Блок 3: Обновление счётчиков в структуре из блока 1

Блок 4: Открытие файла “workers\_register.txt” на перезапись

Блок 5: Проверка успешности открытия файла

Блок 6: Форматированная запись шапки таблицы в файл

Блок 7: Запись данных структуры из блока 1 в файл, по участкам

Блок 8: Проверка наличия рабочих на участке

Блок 9: Расчёт средней заработной платы на участке

Блок 10: Закрытие файла



Рисунок 2.7 – Структурная схема алгоритма функции

strToFormat(char \*str, const int length)

Поблочное описание:

Блок 1: Проверка достижения конца переданной строки

Блок 2: Проверка – является ли элемент символом

Блок 3: Преобразование прописных литер к заглавным

Блок 4: Возвращаемое значение функцией



Рисунок 2.8 – Структурная схема алгоритма функции

checkNumeral(short X, short Y, long int num, int maxDigitCount)

Поблочное описание:

Блок 1: Копирование переданного числа во временную переменную

Блок 2: Проверка количества разрядов в целой части числа

Блок 3: Целочисленное деление на 10

Блок 4: Повторное копирование переданного числа

Блок 5: Бесконечный цикл

Блок 6: Определение положения курсора в консоли для форматного вывода

Блок 7: Считывание нажатия клавиши

Блок 8: Соответствие кодов клавишам:

* 13 – Enter
* 96,241 – ~(тильда), на англ. и русской раскладке клавиатуры соответ.

Блок 9: Соответствие кода 27 клавише ESC. Возврат значения без изменения.

Блок 10: Проверка количества разрядов в числе при нажатии клавиши цифры

Блок 11: Изменения значения числа и счётчика разрядов

Блок 12: Проверка количества разрядов в числе при нажатии клавиши Backspace

Блок 13: Изменение значения числа и количества разрядов



Рисунок 2.9 – Структурная схема алгоритма функции

checkPersonalNumber(int num, list \*top)

Поблочное описание:

Блок 1: Проверка существования списка

Блок 2: Проверка является элемент последним

Блок 3: Проверка переданного номера и номера текущего элемента

Блок 4: Возвращаемое значение при совпадении номеров

Блок 5: Возвращаемое значение при отсутствии совпадения номеров



Рисунок 2.10 – Структурная схема алгоритма функции

saveFile(list \*top, char \*fileName, bool mode)

Поблочное описание:

Блок 1: Проверка существования списка

Блок 2: Создание файловой переменной для работы с потоками

Блок 3: Проверка режима сохранения файла

Блок 4: Установка файла на запись данных в бинарном виде

Блок 5: Проверка успешности открытия файла

Блок 6: Проверка является ли элемент последним в списке

Блок 7: Сохранение структуры в бинарном виде в файл

Блок 8: Закрытие файлового потока

Блок 9: Возвращаемое значение при успешной записи в файл

Блок 10: Установка файла на запись данных в текстовом виде

Блок 11: Проверка успешности открытия файла

Блок 12: Проверка является ли элемент последним в списке

Блок 13: Сохранение каждого поля структуры в текстовый файл

Блок 14: Закрытие файлового потока

Блок 15: Возвращаемое значение при успешной записи в файл

Блок 16: Возвращаемое значение если список оказался пустым

Блок 17: Возвращаемое значение при ошибке во время открытия файла

 Рисунок 2.11 – Структурная схема алгоритма функции loadFile(list \*&top, list \*&end, char \*fileName)

Поблочное описание:

Блок 1: Проверка существования списка

Блок 2: Создание файловой переменной для работы с потоками

Блок 3: Проверка режима загрузки файла

Блок 4: Установка файла на чтение данных в бинарном виде

Блок 5: Проверка успешности открытия файла

Блок 6: Проверка достижения конца файла

Блок 7: Сохранение данных в структуру

Блок 8: Проверка наличия элементов в списке

Блок 9: Вызов функции организации списка

Блок 10: Вызов функции добавления элемента в список

Блок 11: Закрытие файлового потока

Блок 12: Возвращаемое значение при успешной загрузке данных из бинарного файла

Блок 13: Установка файла на чтение данных в текстовом виде

Блок 14: Проверка успешности открытия файла

Блок 15: Проверка достижения конца файла

Блок 16: Сохранение данных в поля структуры

Блок 17: Проверка наличия элементов в списке

Блок 18: Вызов функции организации списка

Блок 19: Вызов функции добавления элемента в список

Блок 20: Закрытие файлового потока

Блок 21: Возвращаемое значение при успешной загрузке данных из текстового файла

Блок 22: Возвращаемое значение при неудачном открытии файла

Блок 23: Возвращаемое значение в случае не пустого списка



Рисунок 2.12 – Структурная схема алгоритма функции

menu(const char \*\*menuItems, const int itemsCount, int currentItem)

Поблочное описание:

Блок 1: Бесконечный цикл

Блок 2: Проверка конца переданного массива

Блок 3: Позиционирование курсора в консоли

Блок 4: Проверка счётчика и номера текущего пункта меню

Блок 5: Выделение текущего пункта меню другим цветом

Блок 6: Вывод пункта меню на экран

Блок 7: Графическое отображение строки состояния

Блок 8: Считывание кода клавиши

Соответствие кодов клавишам:

* 13 – Enter
* 27 – ESC
* 80 – клавиша навигации «вниз»
* 72 – клавиша навигации «вверх»

Блок 9: Возврат функцией номера пункта меню

Блок 10: Возврат функцией кода клавиши ESC означающей выход из подпрограммы

Блок 11: Проверка является пункт меню последним в переданном массиве

Блок 12: Увеличение счётчика пунктов меню

Блок 13: Обнуление счётчика пунктов меню

Блок 14: Проверка является пункт меню первым в переданном массиве

Блок 15: Уменьшение счётчика пунктов меню

Блок 16: Установка счётчика пунктов меню в максимальное значение соответствующее количеству элементов в переданном массиве

1. Обоснование состава технических и программных средств.

Для выполнения задания курсового проектирования был выбран язык С++, так как он является языком программирования общего назначения. Для языка С++, базовым является язык С, поэтому сохранена тесная взаимосвязь между этими языками. В них используются схожие, а иногда и однотипные команды, функции и конструкции, однако язык С++ расширяется введением гибких и эффективных средств, предназначенных для разработки новых типов данных, а так же более широким спектром подключаемых библиотек.

Для экономии оперативной памяти во время работы программы, а так же экономии физической памяти на жестком диске, при сохранении данных в файл, в программе используются типы данных, позволяющие минимизировать расходы памяти.

Размер информационной структуры tableData составляет:

unsigned int personalNumber ; - 4 байта;

unsigned short exp; - 2 байта;

unsigned short rank; - 2 байта;

unsigned short factoryNumber; - 2 байта;

unsigned short deportmentNumber; - 2 байта;

char fio[21]; - 21 байт;

prof[11]; - 11 байт;

unsigned short birth\_year; - 2 байта;

bool sex; - 1 байт;

float salary; - 4 байта;

Итого: 51 байт, но в оперативной памяти будет создаваться динамическая структура list, размер которой равен:

tableData inf; - 51 байта;

struct list \*next; - 8 байт;

struct list \*pred; - 8 байт;

Итого: 67 байт.

Исходя из расчётов, можно сделать вывод, что размер информационной структуры tableData, равный 51 байту, соответствует размеру бинарного файла, при сохранении в него одной записи. Можно отметить, что при сохранении данных в текстовый файл, его размер будет отличаться, так как информация будет сохраняться посимвольно. Учитывая специфику работы программы и введенные ограничения для ввода данных, средний размер текстового файла составит 35 байт.

Размер исходных файлов программы составляет 59,6 кб.

Размер файла запуска программы «CP\_db.exe» составляет 1.9 мб.

# ВЫПОЛНЕНИЕ ПРОГРАММЫ

1. Условия выполнения программы

Скомпилированная программа занимает 1,9 мб физической памяти.

При сохранении базы в файл, его размер занимаемой физической памяти, будет соответствовать произведению количества записей, на объём памяти занимаемой одной записью, в нашем случае 51 байт.

Во время работы программы при создании очередного элемента списка выделяется 67 байт памяти, с учётом специфики работы программы, можно подсчитать, что для корректной работы программы и обработки 107 записей будет достаточно 1024 мб оперативной памяти, что с учетом современных вычислительных мощностей, удовлетворяет потребностям любого предприятия.

Для корректной работы программы выдвигаются требования к аппаратному и программному обеспечению, наличие:

* IBM-совместимого компьютера;
* накопителя на жестком диске;
* достаточное количество оперативной памяти;
* VGA видеоадаптера;
* Установленная операционная система семейства Windows, версии не ниже XP;
* Монитор, с разрешением экрана не менее 640х480 пикселей;

1. Загрузка и запуск программы.

Программа представлена в виде файла с именем CP\_db.exe. Её можно запустить так же, как и любой другой файл имеющий расширение .exe, т.е. используя двойной клик левой кнопкой мыши на иконке программы или выделив иконку одинарным кликом и нажав клавишу Enter.

После запуска программы на экране появится основное меню рис. 3.1.

Текущий пункт меню выделяется бирюзовым цветом, а так же отмечается специальными символами с обеих сторон.

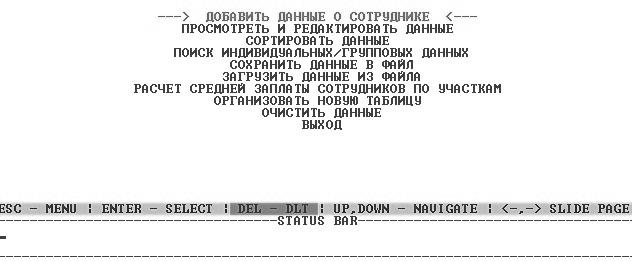


Рисунок 3.1 – Основное меню программы

Для смены пункта меню пользователь может использовать клавиши навигации «↑» и «↓».После установки указателя на нужном пункте меню, пользователь может выбрать действие, которое выполнится после нажатия на клавишу «Enter».

Значение пунктов меню:

* «Добавить данные о сотруднике» - позволяет создать новую запись в базе и ввести данные с клавиатуры, если в базе отсутствуют данные необходимо выбрать пункт меню «Организовать новую таблицу»;
* «Просмотреть и редактировать данные» - отображает на экране текущее состояние базы данных, а так же позволяет редактировать её;
* «Сортировать данные» - выводит на экран подменю содержащее список возможных сортировок базы данных;
* «Поиск индивидуальных/групповых данных» - выводит на экран подменю содержащее список возможных критериев поиска;
* «Сохранить данные в файл» - выводит на экран запрос о сохранении файла, в текущий или новый файл;
* «Загрузить данные из файла» - выводится запрос об очистке текущей базы данных, а после позволяет загрузить данные из текстового или бинарного файла;
* «Расчет средней зарплаты сотрудников по участкам» - производит расчёт средней заработной платы сотрудников на участках, выводит данную информацию в файл «workers\_register.txt”, находящийся в директории с программой;
* «Организовать новую таблицу» - позволяет организовать новую базу данных;
* «Очистить данные» - очищает текущие данные, которые находятся в работе;
* «Выход» - завершает работу программы, если в программе имеются данные, перед выходом выводит запрос об их сохранении, так же выход можно осуществить при помощи клавиши «Esc»;

После запуска программы была произведена загрузка данных из бинарного файла «db1.bin» рис. 3.2:

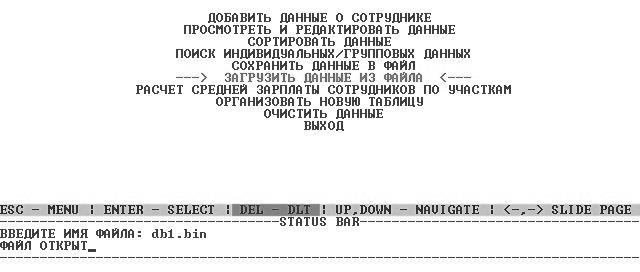


Рисунок 3.2 – Выполнение загрузки данных из файла

Загруженная база имеет вид представленный на рис.3.3 и рис.3.4:

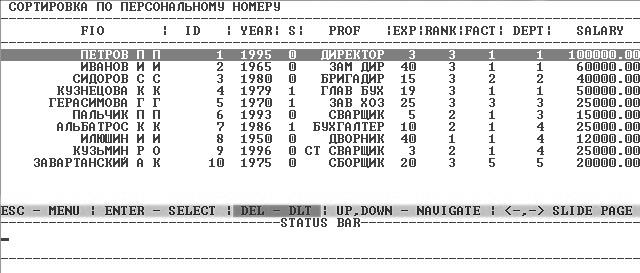


Рисунок 3.3 – Просмотр данных загруженных из базы стр.1

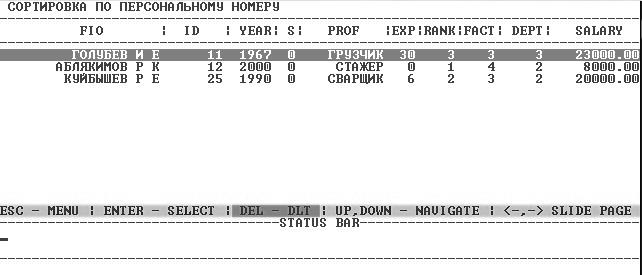


Рисунок 3.4 – Просмотр данных загруженных из базы стр.2

Для создания новой базы вначале необходимо выбрать пункт меню «Очистить данные» (рис.3.5). Таким же при завершении работы программы, происходит очистка памяти неявным вызовом этой функции.

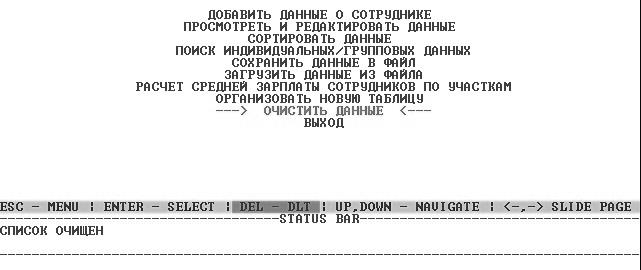


Рисунок 3.5 – Очистка базы данных

После заполнения информационных данных рис.3.6, в базу был добавлен новый элемент рис. 3.7:

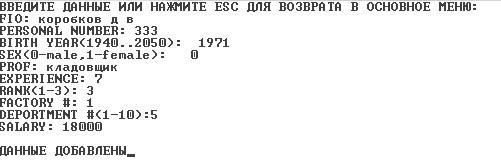


Рисунок 3.6 – Заполнение информационных данных новых работников

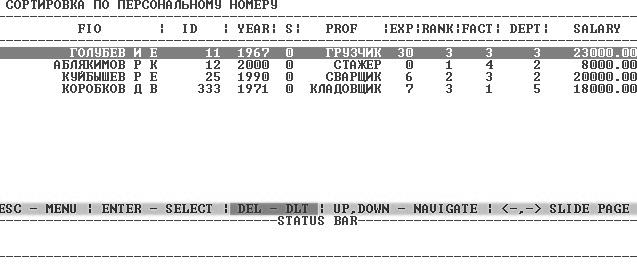


Рисунок 3.7 – Просмотр данных после добавления новой записи

При выборе пункта меню «Сортировать данные», на экране появится дополнительное меню содержащее список возможных сортировок базы данных рис.3.8:

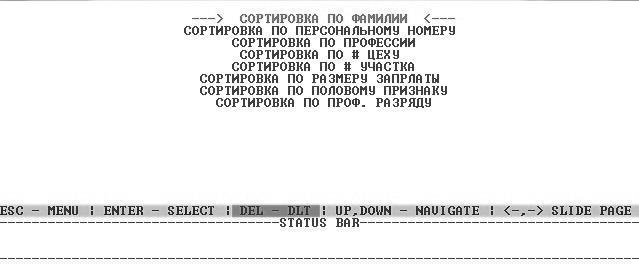


Рисунок 3.8 – Меню со списком сортировок данных в базе

Сортировка производится по убыванию номеров для числовых полей и по алфавиту для буквенных. При отображении данных в таблице, в верхнем левом углу консоли будет отображаться метод сортировки данных.

При выборе пункта меню «Поиск индивидуальных/групповых данных», на экране появится дополнительное меню со списком критериев поиска, рис.3.9:

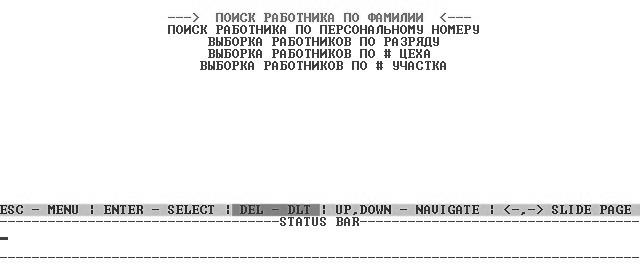


Рисунок 3.9 – Меню со списком вариантов поиска данных в базе

Пункты «выборка» осуществляют поиск группы сотрудников по заданному ключу, и выводит их на экран в виде таблицы. Если программе не удалось найти данные, на экран будет выведено соответствующее сообщение. На рис.3.10 изображен пример отображения найденных данных.

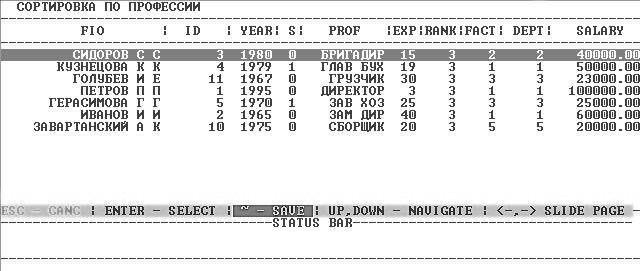


Рисунок 3.10 – Результат поиска сотрудников по разряду

Для удаления записи достаточно выбрать её и нажать на клавишу «Delete» на клавиатуре, после чего будет выведен запрос о подтверждении рис.3.11:

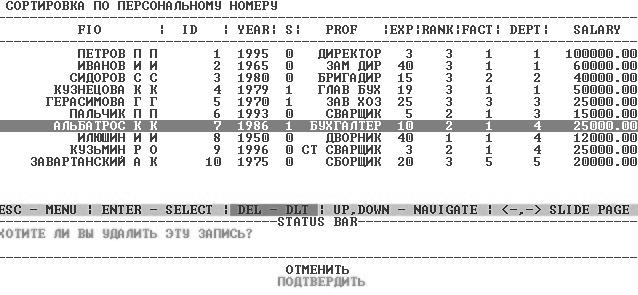


Рисунок 3.11 – Запрос об удалении элемента

Результат удаления приведен на рис.3.12:

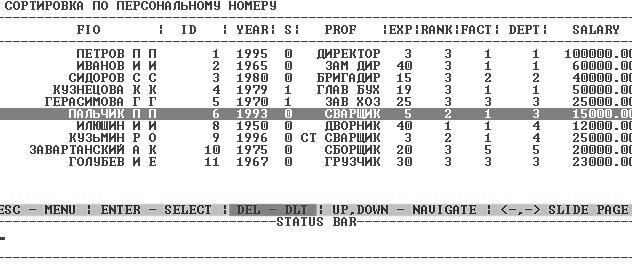


Рисунок 3.12 – Просмотр данных после удаления записи

При попытке пользователя произвести безвозвратные изменения в базе, а так же при сохранении данных в файл будет выведено предупреждающее сообщение рис.3.13, которое ожидает подтверждения или отмены дальнейших действий.

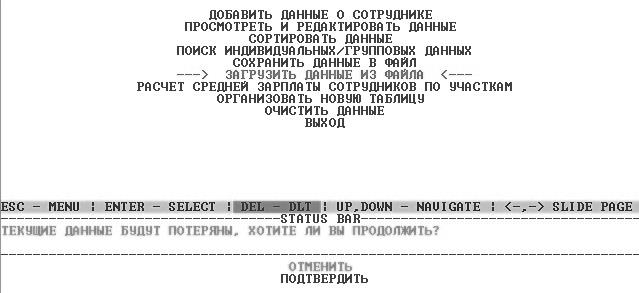


Рисунок 3.13 – Результат поиска сотрудников по разряду

1. Проверка работоспособности программы.

Для проверки работоспособности программы был разработан ряд тестовых примеров.

По умолчанию в программе используется сортировка по табельному номеру. Возможно применение сортировки из списка представленного на рис.3.8. На рис 3.14 представлена сортировка данных по номеру цеха:

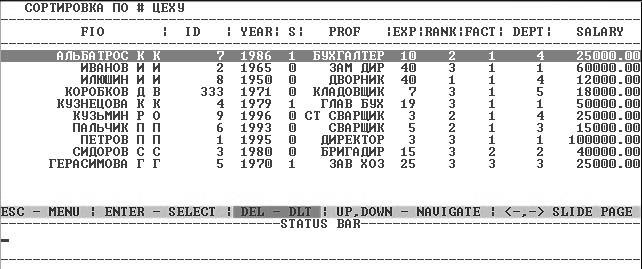


Рисунок 3.14 – Просмотр данных после сортировки данных по номеру цеха

Для проверки работоспособности функции поиска, был выбран соответствующий пункт. В качестве критерия для поиска был выбран пункт меню «Выборка работников по # участка» и введён ключ поиска «2», рис.3.15:

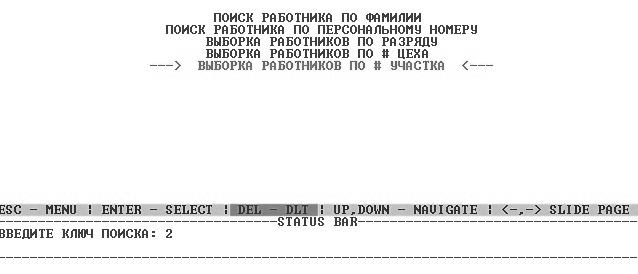


Рисунок 3.15 – Выбор критерия поиска

Результат выборки изображен на рис.3.16:

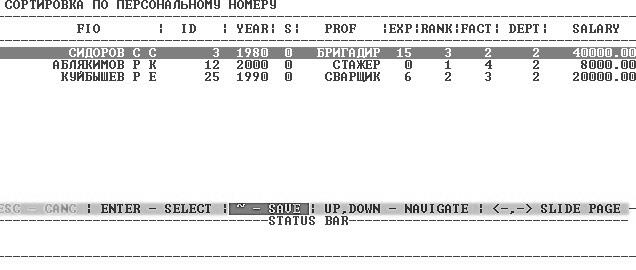


Рисунок 3.16 – Результат выборки из базы данных по участку №2

Для проверки функционала заданного по варианту был выбран пункт меню «Расчет средней зарплаты сотрудников по участкам», результат выполнения отображен на рис.3.17:

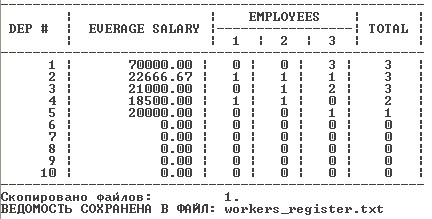


Рисунок 3.17 – Результат расчёта средней заработной платы на участках

# ВЫВОД

В ходе выполнения задания на курсовое проектирование была написана программа, обеспечивающая обработку данных служащих завода и расчёта их средней заработной платы по участкам. В частности, программа обеспечивает первоначальное создание базы данных вручную или с помощью считывания из файла, сохранение ее в текстовый, а также бинарный файлы, с возможностью задания пользователем имени используемого файла. Кроме того, программа предоставляет возможность добавления новых элементов в уже сохраненную базу данных, редактирование любого элемента по любому полю таблицы, удаление любого элемента, поиск, сортировку всех данных по заданному полю, подсчет средней заработной платы по участкам. Были разработаны схемы алгоритмов основных функций, с которыми взаимодействует программа. В основу организации данных программы положена структура в виде двунаправленного динамического списка, позволяющая выполнять просмотр данных в двух направлениях, обеспечивающая легкий доступ к любому элементу, а также рациональное использование оперативной памяти.

Таким образом, поставленная цель была достигнута. Разработанная программа будет полезна для работы на производственных предприятиях, где требуется учёт сотрудников и их заработной платы.

# ПЕРЕЧЕНЬ ССЫЛОК

1. ГОСТ 19.701-90 (ИСО 5807-85) ЕСПД. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения.
2. Методические указания к курсовому проектированию по дисциплине «Основы программирования и алгоритмические языки» для студентов дневной и заочной форм обучения направления 09.03.02 – «Информационные системы и технологии»/ Сост. В.Н. Бондарев, Т.И. Сметанина. – Севастополь: Изд-во СевГУ, 2015. – 72 с.
3. Керниган Б. В., Ритчи Д. М. Язык программирования Си: Пер. с англ. — 3-е изд. — СПб.: Невский Диалект, 2001. — 352 с.
4. Бочков С. О., Субботин Д. М. Язык программирования Си для персонального компьютера. — М.: Радио и связь, 1990. — 384 с.

# Приложение А

(обязательное)

# Текст программы

//Файл “main.cpp”

#include "CP\_types.h"

//MAIN FUNCTION

int main() {

SetConsoleCP(866);

SetConsoleOutputCP(866);

while (1) {

switch (main\_menu\_current\_item = menu(mainMenu, mainMenuItemsCount, main\_menu\_current\_item)) {

case 0: {

if (listHead != NULL) {

if (!addPerson(listEnd, newRecord())) {

printf("\nДАННЫЕ ДОБАВЛЕНЫ");

getch();

} else {

printf("НЕОБХОДИМО ОРАНИЗОВАТЬ ОЧЕРЕДЬ!");

getch();

} break;

}

case 1: {

if (listHead != NULL) {

viewList(listHead, listEnd, 0);

} else {

cleanStatusBar();

emptyMessage();

} break; }

case 2: {

int sortMode = menu(sortMenu, sortMenuItemsCount);

if ((sort\_mode = sort(listHead, listEnd, sortMode)) >= 0) {

viewList(listHead, listEnd, 0);

} break; }

case 3: {

int searchMode = menu(searchMenu, searchMenuItemsCount);

searchData(listHead, searchMode);

getch();

break; }

case 4: {

saveFileInterface();

getch();

break; }

case 5: {

loadFileInterface();

getch();

sort\_mode = sort(listHead, listEnd);

break; }

case 6: {

if (listHead != NULL){

printf("ФАЙЛ ВЕДОМОСТИ БУДЕТ ПЕРЕЗАПИСАН, ПРОДОЛЖИТЬ?");

if (menuInterface(acceptMessage,acceptMessageItemsCount)){

registerOfWorkers(listHead);

} else { printf("ЗАПИСЬ ОТМЕНЕНА");}

} else {

emptyMessage();}getch();

break;

}

case 7: {

if (listHead == NULL) {

if (!organizeList(listHead, listEnd, newRecord())) {

printf("\nДАННЫЕ ДОБАВЛЕНЫ! ОЧЕРЕДЬ ОРГАНИЗОВАНА!");

getch();

}

} else {

printf("\nОЧЕРЕДЬ УЖЕ ОРГАНИЗОВАННА, ИСПОДБЗУЙТЕ ФУНКЦИЮ ДОБАВЛЕНИЯ");

getch();

}

break; }

case 8:{

if (listHead != NULL){

printf("ХОТИТЕ ЛИ ВЫ ОЧИСТИТЬ СПИСОК?");

if (menuInterface(acceptMessage,acceptMessageItemsCount)){

cleanStatusBar();

bool result = deleteList(listHead);

if(result) {

printf("СПИСОК ОЧИЩЕН");

} else {

emptyMessage();

}

} else {

cleanStatusBar();

printf("ОТМЕНЕНО");

}

} else {

emptyMessage();

}

getch();break; }

case 9:

case 27: {

printf("ВЫЙТИ?");

if (listHead != NULL) {

unsigned exitChange = menuInterface(exitMessage, exitMessageItemsCount);

if (exitChange == 0) {

cleanStatusBar();

SetColor(12, 0);

printf("ПЕРЕЗАПИСАТЬ ФАЙЛ?:");

SetColor(7, 0);

if (menuInterface(acceptMessage, acceptMessageItemsCount)) {

cleanStatusBar();

if (!saveFile(listHead, openFileName)) {

printf("ФАЙЛ СОХРАНЕН");

} else {

printf("ФАЙЛ НЕ СОХРАНЕН!");

} getch();} }

deleteList(listHead);

exit(0); } else {

if (menuInterface(acceptMessage, acceptMessageItemsCount) == 1) {

deleteList(listHead);

exit(0); } break; } }}}}

-------------------------------------------------------------------------------------------------------

//Файл “CP\_types.h”

#if !defined(CP\_types)

#define CP\_types

#include <conio.h>

#include <iostream>

#include <fstream>

#include <windows.h>

#include <iomanip>

using namespace std;

/\*

\* STRUCTURES AND TYPES

\*/

const int mainMenuItemsCount = 10;

const char \*mainMenu[mainMenuItemsCount] = {

"ДОБАВИТЬ ДАННЫЕ О СОТРУДНИКЕ",

"ПРОСМОТРЕТЬ И РЕДАКТИРОВАТЬ ДАННЫЕ",

"СОРТИРОВАТЬ ДАННЫЕ",

"ПОИСК ИНДИВИДУАЛЬНЫХ/ГРУППОВЫХ ДАННЫХ",

"СОХРАНИТЬ ДАННЫE В ФАЙЛ",

"ЗАГРУЗИТЬ ДАННЫЕ ИЗ ФАЙЛА",

"РАСЧЕТ СРЕДНЕЙ ЗАРПЛАТЫ СОТРУДНИКОВ ПО УЧАСТКАМ",

"ОРГАНИЗОВАТЬ НОВУЮ ТАБЛИЦУ",

"ОЧИСТИТЬ ДАННЫЕ",

"ВЫХОД"

};

const int acceptMessageItemsCount = 2;

const char \*acceptMessage[acceptMessageItemsCount] = {

"ОТМЕНИТЬ",

"ПОДТВЕРДИТЬ"

};

const int saveFileMessageItemsCount = 2;

const char \*saveFileMessage[saveFileMessageItemsCount] = {

"СОХРАНИТЬ В ТЕКЩИЙ ФАЙЛ",

"СОХРАНИТЬ В ДРУГОЙ ФАЙЛ"

};

const int exitMessageItemsCount = 2;

const char \*exitMessage[exitMessageItemsCount] = {

"СОХРАНИТЬ И ВЫЙТИ",

"ВЫЙТИ БЕЗ СОХРАНЕНИЯ"

};

const int searchMenuItemsCount = 5;

const char \*searchMenu[searchMenuItemsCount] = {

"ПОИСК РАБОТНИКА ПО ФАМИЛИИ",

"ПОИСК РАБОТНИКА ПО ПЕРСОНАЛЬНОМУ НОМЕРУ",

"ВЫБОРКА РАБОТНИКОВ ПО РАЗРЯДУ",

"ВЫБОРКА РАБОТНИКОВ ПО # ЦЕХА",

"ВЫБОРКА РАБОТНИКОВ ПО # УЧАСТКА"

};

const int sortMenuItemsCount = 8;

const char \*sortMenu[sortMenuItemsCount] = {

"СОРТИРОВКА ПО ФАМИЛИИ",

"СОРТИРОВКА ПО ПЕРСОНАЛЬНОМУ НОМЕРУ",

"СОРТИРОВКА ПО ПРОФЕССИИ",

"СОРТИРОВКА ПО # ЦЕХУ",

"СОРТИРОВКА ПО # УЧАСТКА",

"СОРТИРОВКА ПО РАЗМЕРУ ЗАПРЛАТЫ",

"СОРТИРОВКА ПО ПОЛОВОМУ ПРИЗНАКУ",

"СОРТИРОВКА ПО ПРОФ. РАЗРЯДУ"

};

const unsigned short int deportment\_count = 10;

const unsigned short int rank\_count = 3;

const int FIO\_LENGTH = 21;

const int PROF\_LENGTH = 11;

const int MAX\_STR\_LENGTH = 256;

unsigned console\_row\_length = 80;

char openFileName[MAX\_STR\_LENGTH];

const unsigned countOfDisplayRecords = 10;

int key, main\_menu\_current\_item = 0;

short int sort\_mode = 1;

struct tableData {

unsigned int personalNumber;

char fio[FIO\_LENGTH];

unsigned short birth\_year;

bool sex;

char prof[PROF\_LENGTH];

unsigned short exp;

unsigned short rank;

unsigned short factoryNumber;

unsigned short deportmentNumber;

float salary;

};

const int TABLE\_DATA\_SIZE = sizeof(tableData);

struct list {

tableData inf;

struct list \*next, \*pred;

};

list \*listHead = NULL, \*listEnd = NULL;

//FUNCTIONS PROTOTYPES

//CONSOLE MANIPULATION

void gotoxy(short x, short y);

void SetColor(int text, int background);

void readTheKey();

//FILE

int saveFile(list \*top, char \*fileName, bool mode = 1);

int loadFile(list \*&top, list \*&end, char \*fileName);

void saveFileInterface();

void loadFileInterface();

//LIST MANIPULATION

tableData newRecord();

int organizeList(list \*&top, list \*&end, tableData personalData);

int addPerson(list \*&end, tableData personalData);

tableData editData(tableData current);

int deleteList(list \*&top);

int deletePersonalData(list \*&listHead, list \*&listEnd, list \*current);

void viewList(list \*&listHead, list \*&end, unsigned mode = 0);

//INTERFACE

void outData(list \*temp);

int menu(const char \*\*menuItems, const int itemsCount, int currentItem = 0);

int menuInterface(const char \*\*menuItems, const int itemsCount = 2);

void drawHelpMenu(unsigned mode = 0);

void drawTableHead(unsigned X = 0, unsigned Y = 0);

void cleanPlace();

void cleanStatusBar();

void emptyMessage();

//CHECK AND FORMAT

unsigned int checkNumeral(short X = 0, short Y = 0, long int num = 0, int maxDigitCount = 2);

char \*strToFormat(char \*str, const int length);

int rewriteString(unsigned X, unsigned Y, const int length, char \*tempStr);

unsigned checkPersonalNumber(int num, list \*top);

//SORT

int sort(list \*&head, list \*&end, short int mode = 1);

int searchData(list \*&head, short int mode = 0);

void registerOfWorkers(list \*head);

#include "CP\_consmanip.cpp"

#include "CP\_listmanip.cpp"

#include "CP\_filemanip.cpp"

#include "CP\_sort.cpp"

#include "CP\_interface.cpp"

#include "CP\_checks.cpp"

#endif

-------------------------------------------------------------------------------------------------------

//Файл “CP\_checks.cpp”

#include "CP\_types.h"

/\*

\* ФУНКЦИЯ ПРОВЕРКИ ЧИСЛА НА ВВОД, С ВОЗМЛЖНОСТЬЮ ВЫБОРА КООРДИНАТ

\* ПРИ ПЕРЕДАЧЕ X < 0 И Y < 0 - КООРДИНАТЫ НЕ УЧИТЫВАЮТСЯ

\* ПРИ НАЖАТИИ ESC - ВОЗРАЩАЕТСЯ 0, И НАЧАЛЬНОЕ ЗНАЧЕНИЕ ПЕРЕДАННОЕ ФУНКЦИИ ИЛИ 0

\*/

unsigned int checkNumeral(short X, short Y, long int num, int maxDigitCount) {

long int tempNum = num;

int currentDigitCount, minDigitCount = 0;

for (currentDigitCount = 0; tempNum >= 1; currentDigitCount++) {

tempNum = tempNum / 10; }

tempNum = num;

while (1) {

if ((X >= 0) && (Y >= 0)) {

gotoxy(X, Y); }

for (int i = 0; i < maxDigitCount + 1; i++) {

putch(' '); }

if ((tempNum) || (maxDigitCount == 1)) {

if ((X >= 0) && (Y >= 0)) {

gotoxy(X, Y); }

printf("%d \n", tempNum); }

switch (key = getch()) {

case '0': case '1': case '2': case '3':

case '4': case '5': case '6': case '7': case '8': case '9': {

if (currentDigitCount < maxDigitCount) {

tempNum = tempNum \* 10 + (key - '0');

currentDigitCount++;

}

break; }

case 8: {

if (currentDigitCount > minDigitCount) {

tempNum = tempNum / 10;

currentDigitCount--;

}

break;

}

case 13:

case 96:

case 241: {

return tempNum;

}

case 27: {

return num;

} } }

}

/\*

\* ПРИВЕДЕНИЕ СИМВОЛОВ ФАМИЛИИ И ИНИЦИАЛОВ К ВЕРХНЕМУ РЕГИСТРУ(УЧИТЫВАЯ РУССКИЕ СИМВОЛЫ)

\*/

char \*strToFormat(char \*str, const int length) {

int i;

int k;

for (i = 0; (i < length) && (str[i] != '\0'); i++) {

if ((isspace(str[i])) || (ispunct(str[i]))) { //если i-тый элемент пробел или знак - пропуск

continue;

} else {

k = str[i];

if ((k >= -96) && (k <= -81)) {

str[i] = k - 32;

} else if ((k >= -32) && (k <= 0)) {

str[i] = k - 80;

} else {

str[i] = toupper(str[i]);

} } }

return str;

}

/\*

\* ФУНКЦИЯ ПРОВЕРКИ СИМВОЛОВ НА ВВОД, С ВОЗМЛЖНОСТЬЮ ВЫБОРА КООРДИНАТ

\* ПРИ ПЕРЕДАЧЕ X<0 И Y <0 - КООРДИНАТЫ НЕ УЧИТЫВАЮТСЯ

\* ПРИ НАЖАТИИ ESC - ВОЗРАЩАЕТСЯ 0 И НАЧАЛЬНОЕ ЗНАЧЕНИЕ СТРОКИ НЕ ИЗМЕНЯЕТСЯ

\*/

int rewriteString(unsigned X, unsigned Y, const int length, char \*str) {

char tempStr[length];

strcpy(tempStr, str);

unsigned currentPosition;

currentPosition = strlen(str);

while (1) {

if ((X >= 0) && (Y >= 0)) {

gotoxy(X, Y);

}

for (int i = 0; i < length - 1; i++) {

putch(' ');

}

if ((X >= 0) && (Y >= 0)) {

gotoxy(X, Y);

}

cout << tempStr << endl;

switch (key = getch()) {

case 13:

case 96:

case 241: {

strToFormat(tempStr, length);

strcpy(str, tempStr);

return 1;

}

case 8: {

if (currentPosition > 0) {

currentPosition--;

}

break; }

case 27: {

return 0; }

default: {

if ((((key >= 65) && (key <= 90)) ||

((key >= 97) && (key <= 122)) ||

((key >= 128) && (key <= 175)) ||

((key >= 224) && (key <= 241)) ||

(key == 32))

&& (currentPosition < length - 1)) {

tempStr[currentPosition++] = (char) key;

} break; } }

tempStr[currentPosition] = '\0'; }}

/\*

\* ФУНКЦИЯ ПРОВЕРКИ СОВПАДЕНИЯ ИНДИВИДУАЛЬНОГО НОМЕРА ПРИ ВВОДЕ ДАННЫХ

\* ВОЗВРАЩАЕТ 1 - ПРИ СОВПАДЕНИИ НОМЕРОВ, 0 - В ПРОТИВНОМ СЛУЧАЕ

\*/

unsigned checkPersonalNumber(int num, list \*top) {

if (top != NULL) {

list \*temp;

for (temp = top; temp != NULL; temp = temp->next) {

if (temp->inf.personalNumber == num) {

return 1;

} }} return 0;}

-------------------------------------------------------------------------------------------------------

//Файл “CP\_consmanip.cpp”

#include "CP\_types.h"

/\*Функция позиционирования курсора в консоли\*/

void gotoxy(short x, short y) {

HANDLE StdOut = GetStdHandle(STD\_OUTPUT\_HANDLE);

COORD coord = {x, y};

SetConsoleCursorPosition(StdOut, coord);}

/\*Функция выбора цвета текста и фона в консоли\*/

void SetColor(int text, int background) {

HANDLE hStdOut = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(hStdOut, (WORD) ((background << 4) | text));}

/\*

\* Функция проверки кода клавиш

\*/

void readTheKey() {

char key2;

while (int key = getch()) {

key2 = key;

cout << key << " = " << key2 << endl;

if (key == 13) {

printf("КОНЕЦ");

return; } }}

-------------------------------------------------------------------------------------------------------

//Файл “CP\_filemanip.cpp”

#include "CP\_types.h"

/\*

\* ФУНКЦИЯ СОХРАНЕНИЯ ОЧЕРЕДИ В ФАЙЛ

\* mode:

\* 0 - текстовый файл или если файл имеет расширение ".txt"

\* ПОСЛЕ ТЕКСТОВЫХ ПОЛЕЙ ДОБАВЛЯЕТ СИМВОЛ '\n'

\* ПОСЛЕ ЧИСЛОВЫХ " "

\* 1 - бинарный файл или если файл имеет расширение ".bin"

\*/

int saveFile(list \*top, char \*fileName, bool mode) {

if (top != NULL) {

list \*temp;

ofstream inFile;

int j;

if ((strstr(fileName, ".txt")) || (mode == 0)) {

inFile.open(fileName, ios::out);

if (inFile) {

for (temp = top; temp != NULL; temp = temp->next) {

inFile << temp->inf.fio << '\n';

inFile << temp->inf.personalNumber << " ";

inFile << temp->inf.birth\_year << " ";

inFile << temp->inf.sex;

inFile << temp->inf.prof << '\n';

inFile << temp->inf.exp << " ";

inFile << temp->inf.rank << " ";

inFile << temp->inf.factoryNumber << " ";

inFile << temp->inf.deportmentNumber << " ";

inFile << (long int) temp->inf.salary;

}

inFile.close();

return 0;

} else { return 1; }

} else if ((strstr(fileName, ".bin")) || (mode == 1)) {

inFile.open(fileName, ios::out | ios::binary);

if (inFile) {

for (temp = top; temp != NULL; temp = temp->next) {

inFile.write((char \*) &temp->inf, TABLE\_DATA\_SIZE);

}

inFile.close();

return 0;

} else { return 1; } }

} else { return 1; }}

/\*

\* ФУНКЦИЯ ЗАГРУЗКИ ОЧЕРЕДИ ИЗ ФАЙЛА

\*/

int loadFile(list \*&top, list \*&end, char \*fileName) {

if (top == NULL) {

ifstream outFile;

if (strstr(fileName, ".txt") != NULL) {

outFile.open(fileName, ios::in);

if (outFile) {

tableData tempData;

int i = 0;

while (!outFile.eof()) {

outFile.getline(tempData.fio, FIO\_LENGTH);////

outFile >> tempData.personalNumber;

outFile >> tempData.birth\_year;

outFile >> tempData.sex;

outFile.getline(tempData.prof, PROF\_LENGTH);/////

outFile >> tempData.exp;

outFile >> tempData.rank;

outFile >> tempData.factoryNumber;

outFile >> tempData.deportmentNumber;

outFile >> tempData.salary;

if (top == NULL) {

organizeList(top, end, tempData);

} else {

addPerson(end, tempData);

} }

outFile.close();

return 0;

} else { return -1; }

} else {

outFile.open(fileName, ios::in | ios::binary);

if (outFile) {

tableData tempData;

while (outFile.read((char \*) &tempData, TABLE\_DATA\_SIZE)) {

if (top == NULL) {

organizeList(top, end, tempData);

} else {

addPerson(end, tempData);

} }

outFile.close();

return 0;

} else { return -1; } }

} else { return 1; }}

/\*

\* ФУНЦИЯ ДЛЯ ВЫЗОВА ФУНКЦИИ ЗАПИСИ В ФАЙЛ

\*/

void saveFileInterface() {

cleanStatusBar();

if (listHead != NULL) {

unsigned saveChange = menuInterface(saveFileMessage, saveFileMessageItemsCount);

if ((!saveChange) && (strlen(openFileName) > 1)) {

cleanStatusBar();

SetColor(12, 0);

printf("ПЕРЕЗАПИСАТЬ?");

SetColor(7, 0);

if (menuInterface(acceptMessage, acceptMessageItemsCount)) {

cleanStatusBar();

if (!saveFile(listHead, openFileName)) {

printf("ФАЙЛ СОХРАНЕН");

} else {

printf("ФАЙЛ НЕ СОХРАНЕН!"); } }

} else if ((saveChange == 1) || strlen(openFileName) < 2) {

cleanStatusBar();

if (strlen(openFileName) < 3) {

printf("ФАЙЛ БУДЕТ СОЗДАН \n");

}

char newFileName[MAX\_STR\_LENGTH];

printf("ВВЕДИТЕ ИМЯ ФАЙЛА(\*.bin - binary, \*.txt - text): ");

cin.getline(newFileName, MAX\_STR\_LENGTH);

if (cin.fail()) { //ПРИ ПЕРЕПОЛНЕНИИ БУФЕРА ВХОДНОГО ПОТОКА

cin.clear(); //СБРОС ОШИБКИ ПОТОКА

cin.ignore(1000, '\n'); //ИГНОРИРОВАНИЕ ОСТАВШИХСЯ В ПОТОКЕ СИМВОЛОВ

}

cleanStatusBar();

if (!saveFile(listHead, newFileName)) {

printf("ФАЙЛ СОХРАНЕН");

strcpy(openFileName, newFileName);

} else {

printf("ФАЙЛ НЕ СОХРАНЕН!");

}

} else if (saveChange == 27) {

return; }

} else {

cleanStatusBar();

emptyMessage();

}}

/\*

\* ФУНЦИЯ ДЛЯ ВЫЗОВА ФУНКЦИИ ЗАКГРУЗКИ ФАЙЛА

\*/

void loadFileInterface() {

SetColor(12, 0);

printf("ТЕКУЩИЕ ДАННЫЕ БУДУТ ПОТЕРЯНЫ, ХОТИТЕ ЛИ ВЫ ПРОДОЛЖИТЬ?");

SetColor(7, 0);

unsigned loadChange = menuInterface(acceptMessage, acceptMessageItemsCount);

if (loadChange == 1) {

deleteList(listHead);

cleanStatusBar();

printf("ВВЕДИТЕ ИМЯ ФАЙЛА: ");

cin.getline(openFileName, MAX\_STR\_LENGTH);

if (cin.fail()) { //ПРИ ПЕРЕПОЛНЕНИИ БУФЕРА ВХОДНОГО ПОТОКА

cin.clear(); //СБРОС ОШИБКИ ПОТОКА

cin.ignore(1000, '\n'); //ИГНОРИРОВАНИЕ ОСТАВШИХСЯ В ПОТОКЕ СИМВОЛОВ

}

if (!loadFile(listHead, listEnd, openFileName)) {

printf("ФАЙЛ ОТКРЫТ");

} else {

printf("НЕ УДАЛОСЬ ОТКРЫТЬ ФАЙЛ!");

} }};

-------------------------------------------------------------------------------------------------------

//Файл “CP\_interface.cpp”

#include "CP\_types.h"

/\*

\* ФУНКЦИЯ ВЫВОДА ДАННЫХ В КОНСОЛЬ(В ОДНУ СТРОКУ)

\*/

void outData(list \*temp) {

cout << setw(20) << temp->inf.fio << " "

<< setw(7) << temp->inf.personalNumber << " "

<< setw(5) << temp->inf.birth\_year << " "

<< setw(2) << temp->inf.sex << " "

<< setw(10) << temp->inf.prof << " "

<< setw(3) << temp->inf.exp << " "

<< setw(4) << temp->inf.rank << " "

<< setw(4) << temp->inf.factoryNumber << " "

<< setw(5) << temp->inf.deportmentNumber << " "

<< setprecision(2) << fixed << setw(11) << temp->inf.salary;}

/\*

\* ФУНКЦИЯ ВЫВОДА И НАВИГАЦИИ ОСНОВНОГО МЕНЮ

\* ВОЗВРАЩАЕТ НОМЕР ПУНКТА МЕНЮ В ПЕРЕДАННОМ МАССИВЕ

\*/

int menu(const char \*\*menuItems, const int itemsCount, int currentItem) {

int i = 0;

int centerCoorOfConsole = 40, coordX = 0, coordY = 0;

while (1) {

system("cls");

for (i = 0; i < itemsCount; i++) {

coordX = centerCoorOfConsole - (strlen(menuItems[i]) / 2);

coordY = i + 1;

if (i == currentItem) {

coordX -= 6;

gotoxy(coordX, coordY);

//SetColor(0, 8);

SetColor(11, 0);

printf("---> ");

} else { gotoxy(coordX, coordY); }

printf("%s", menuItems[i]);

if (i == currentItem) { printf(" <---"); }

printf("\n");

SetColor(7, 0); }

drawHelpMenu();

switch (key = getch()) {

case 13: {

return currentItem;

}

case 80: {

if (currentItem >= itemsCount - 1) currentItem = 0;

else currentItem++;

break;

}

case 72: {

if (currentItem <= 0) currentItem = itemsCount - 1;

else currentItem--;

break;

}

case 27: {

return 27;

} } }}

/\*

\* ФУНКЦИЯ ВЫВОДА И НАВИГАЦИИ ДОПОЛНИТЕЛЬНОГО МЕНЮ(СОХРАНЕНИЯ В ФАЙЛ ИЛИ ПОДТВЕРЖДЕНИЯ ОПЕРАЦИЙ)

\* ВОЗВРАЩАЕТ НОМЕР ПУНКТА МЕНЮ В ПЕРЕДАННОМ МАССИВЕ

\*/

int menuInterface(const char \*\*menuItems, const int itemsCount) {

int currentItem = 0, i = 0;

while (1) {

cleanPlace();

gotoxy((40 - (strlen(menuItems[0]) / 2)), 22);

for (i = 0; i < itemsCount; i++) {

if (i == currentItem) {

SetColor(12, 0);

}

printf("%s \n", menuItems[i]);

SetColor(7, 0);

gotoxy((40 - (strlen(menuItems[1]) / 2)), 23); }

switch (key = getch()) {

case 13: {

cleanPlace();

return currentItem;

}

case 80: {

if (currentItem >= itemsCount - 1) currentItem = 0;

else currentItem++;

break;

}

case 72: {

if (currentItem <= 0) currentItem = itemsCount - 1;

else currentItem--;

break;

}

case 27: {

cleanPlace();

return 27;

} } }}

/\*

\* ФУНКЦИЯ ПЕЧАТИ ДОПОЛНИТЕЛЬНОГО МЕНЮ

\* mode:

\* 0 - выводит пункт "DEL - DLT";

\* 1 - ыводит пункт " ~ - SAVE";

\*/

void drawHelpMenu(unsigned mode) {

gotoxy(0, 17);

SetColor(7, 1);

(!mode) ?

(SetColor(7, 1), printf("ESC - MENU "), SetColor(7, 1)) :

(SetColor(12, 1), printf("ESC - CANC "), SetColor(7, 1));

printf("| ENTER - SELECT |");

(!mode) ?

(SetColor(7, 12), printf(" DEL - DLT "), SetColor(7, 1)) :

(SetColor(9, 14), printf(" ~ - SAVE "), SetColor(7, 1));

printf("| UP,DOWN - NAVIGATE | <-,-> SLIDE PAGE ");

SetColor(7, 0);

for (int i = 0; i < 35; i++) putch('-');

cout << "STATUS BAR";

for (int i = 0; i < 35; i++) putch('-');

gotoxy(0, 21);

for (int i = 0; i < console\_row\_length; i++) putch('-');

cleanStatusBar();

}

/\*

\* ФУНКЦИЯ ВЫВОДА ШАПКИ ТАБЛИЦЫ

\*/

void drawTableHead(unsigned X, unsigned Y) {

gotoxy(X, Y);

printf("%s \n", sortMenu[sort\_mode]);

for (int i = 0; i < console\_row\_length; i++) {

putch('-'); }

printf(" FIO |");

printf(" ID |");

printf(" YEAR|");

printf(" S|");

printf(" PROF |");

printf("EXP|");

printf("RANK|");

printf("FACT|");

printf(" DEPT|");

printf(" SALARY ");

for (int i = 0; i < console\_row\_length; i++) {

putch('-');

}};

/\*

\* ФУНКЦИЯ ОЧИСТКИ КОНСОЛИ НИЖЕ STATUS BAR

\*/

void cleanPlace() {

int i;

gotoxy(0, 22);

for (i = 0; (i < (console\_row\_length \* 2)); i++) putch(' ');

return;

}

/\*

\* ФУНКЦИЯ ОЧИСТКИ STATUS BAR

\*/

void cleanStatusBar() {

int i;

gotoxy(0, 19);

for (i = 0; (i < (console\_row\_length \* 2)); i++) putch(' ');

gotoxy(0, 19);

return;

}

/\*

\* ФУНКЦИЯ ИНДИКАТОРА В СЛУЧАЕ ПУСТОГО СПИСКА

\*/

void emptyMessage() {

SetColor(14, 0);

printf("СПИСОК ПУСТ \n");

SetColor(7, 0);

getch();

}

-------------------------------------------------------------------------------------------------------

//Файл “CP\_listmanip.cpp”

#include "CP\_types.h"

/\*

\* ФУНКЦИЯ ДЛЯ ЗАПОЛНЕНИЯ ДАННЫХ СТРУКТУРЫ ЭЛЕМЕНТА

\* ВОЗВРАЩАЕТ ЗАПОЛНЕННУЮ СТРУКТУРУ

\* ЕСЛИ ДАННЫЕ ПОЛЯ НЕ ВВЕДЕНЫ:

ТЕКСТОВОЕ == "\0":

ЧИСЛОВОЕ == 0;

\*/

tableData newRecord() {

tableData newElement;

system("cls");

printf("ВВЕДИТЕ ДАННЫЕ ИЛИ НАЖМИТЕ ESC ДЛЯ ВОЗВРАТА В ОСНОВНОЕ МЕНЮ: \n");

if (getch() == 27) {

newElement.personalNumber = -1;

return newElement;

}

unsigned coordY = 1, coordX = 5;

printf("FIO: ");

char fio[FIO\_LENGTH] = "\0";

rewriteString(coordX, coordY, FIO\_LENGTH, fio);

strcpy(newElement.fio, fio);

coordY++;

gotoxy(0, coordY);

printf("PERSONAL NUMBER: ");

coordX = 17;

int personal\_number;

bool checkResult = false;

do {

if (checkResult) {

gotoxy(coordX + 11, coordY);

printf("ЗАПИСЬ С ТАКИМИ НОМЕРОМ УЖЕ СУЩЕСТВУЕТ!");

getch();

}

personal\_number = checkNumeral(coordX, coordY, 0, 6);

} while (checkResult = checkPersonalNumber(personal\_number, listHead));

gotoxy(coordX + 11, coordY);

printf(" ");

newElement.personalNumber = personal\_number;

coordY++;

gotoxy(0, coordY);

printf("BIRTH YEAR(1940..2050): ");

coordX = 25;

unsigned year = 0;

while ((year < 1940) || (year > 2050)) {

year = checkNumeral(coordX, coordY, 0, 4);

}

newElement.birth\_year = year;

coordY++;

gotoxy(0, coordY);

printf("SEX(0-male,1-female): ");

coordX = 24;

short int sex = -1;

while ((sex < 0) || (sex > 1)) {

sex = checkNumeral(coordX, coordY, 0, 1);

}

newElement.sex = sex;

coordY++;

gotoxy(0, coordY);

printf("PROF: ");

char prof[PROF\_LENGTH] = "\0";

coordX = 6;

rewriteString(coordX, coordY, PROF\_LENGTH, prof);

strcpy(newElement.prof, prof);

coordY++;

gotoxy(0, coordY);

printf("EXPERIENCE: ");

coordX = 12;

newElement.exp = checkNumeral(coordX, coordY, 0);

coordY++;

gotoxy(0, coordY);

printf("RANK(1-%d): ", rank\_count);

coordX = 11;

short int rank = 0;

while ((rank < 1) || (rank > 3)) {

rank = checkNumeral(coordX, coordY, 0, 1);

}

newElement.rank = rank;

coordY++;

gotoxy(0, coordY);

printf("FACTORY #: ");

coordX = 11;

newElement.factoryNumber = checkNumeral(coordX, coordY, 0);

coordY++;

gotoxy(0, coordY);

printf("DEPORTMENT #(1-%d): ", deportment\_count);

coordX = 19;

short int deportment = 0;

while ((deportment < 1) || (deportment > 10)) {

deportment = checkNumeral(coordX, coordY, 0, 2);

}

newElement.deportmentNumber = deportment;

coordY++;

gotoxy(0, coordY);

printf("SALARY: ");

coordX = 8;

newElement.salary = (float) checkNumeral(coordX, coordY, 0, 7);

return newElement;

}

/\*

\* ФУНКЦИЯ ОРГАНИЗАЦИИ ОЧЕРЕДИ

\* ПОЗВОЛЯЕТ ДОБАВИТЬ ТОЛЬКО ПЕРВЫЙ ЭЛЕМЕНТ

\* ВОЗВРАЩАЕТ:

\* 0 - В СЛУЧАЕ ДОБАВЛЕНИЯ

\* 1 - В СЛУЧАЕ НЕ ПУСТОГО СПИСКА

\*/

int organizeList(list \*&top, list \*&end, tableData personalData) {

if ((top == NULL) && (personalData.personalNumber != -1)) {

struct list \*newAdress = new list;

newAdress->inf = personalData;

newAdress->next = NULL;

newAdress->pred = NULL;

end = top = newAdress;

return 0;

}

return 1;

}

/\*

\* ФУНКЦИЯ ДОБАВЛЕНИЯ ЭЛЕМЕНТА В ОЧЕРЕДЬ

\* ДОБАВЛЯЕТ ЭЛЕМЕНТЫ ТОЛЬКО, ЕСЛИ СПИСОК НЕ ПУСТ

\* ВОЗВРАЩАЕТ:

\* 0 - В СЛУЧАЕ ДОБАВЛЕНИЯ

\* 1 - В СЛУЧАЕ ПУСТОГО СПИСКА

\*/

int addPerson(list \*&end, tableData personalData) {

if ((end != NULL) && (personalData.personalNumber != -1)) {

list \*New = new list;

New->inf = personalData;

New->next = NULL;

New->pred = end;

end->next = New;

end = New;

return 0;

} return 1;}

/\*

\* ФУНКЦИЯ РЕДАКТИРОВАНИЕ ЗАПИСИ

\* ПРИ НАЖАТИИ: ~ - ВОЗВРАЩАЕТ ИЗМЕНЁННУЮ СТРУКТУРУ

\* ESC - СТРУКТУРУ БЕЗ ИЗМЕНЕНИЙ

\*/

tableData editData(tableData mainData) {

unsigned countOfFields = 10;

unsigned currentField = 0, i;

unsigned coordX = 0, coordY = 19;

tableData current = mainData;

drawHelpMenu(1);

int temp\_number = current.personalNumber;

while (1) {

cleanStatusBar();

for (i = 0; (i < countOfFields); i++) {

if (i == currentField) {

SetColor(0, 8);

}

switch (i) {

case 0: {

cout << setw(20) << current.fio << " ";

break; }

case 1: {

cout << setw(7) << current.personalNumber << " ";

break; }

case 2: {

cout << setw(5) << current.birth\_year << " ";

break; }

case 3: {

cout << setw(2) << current.sex << " ";

break; }

case 4: {

cout << setw(10) << current.prof << " ";

break; }

case 5: {

cout << setw(3) << current.exp << " ";

break; }

case 6: {

cout << setw(4) << current.rank << " ";

break; }

case 7: {

cout << setw(4) << current.factoryNumber << " ";

break; }

case 8: {

cout << setw(5) << current.deportmentNumber << " ";

break; }

case 9: {

cout << setprecision(2) << fixed << setw(11) << current.salary;

break; } }

SetColor(7, 0); }

switch (key = getch()) {

case 13: {

switch (currentField) {

case 0: {

coordX = 0;

rewriteString(coordX, coordY, FIO\_LENGTH, current.fio);

break;

}

case 1: {

coordX = 22;

//

bool checkResult = false;

do {

if (checkResult){

SetColor(4,0);

}

current.personalNumber = checkNumeral(coordX, coordY, current.personalNumber, 6);

SetColor(7,0);

} while ((temp\_number != current.personalNumber) && (checkResult = checkPersonalNumber(current.personalNumber, listHead)));

//

//current.personalNumber = checkNumeral(coordX, coordY, current.personalNumber, 6);

break;

}

case 2: {

coordX = 30;

do{

if((current.birth\_year < 1940) || (current.birth\_year > 2050)) {

SetColor(4,0);

}

current.birth\_year = checkNumeral(coordX, coordY, current.birth\_year, 4);

SetColor(7,0);

} while ((current.birth\_year < 1940) || (current.birth\_year > 2050));

break;

}

case 3: {

coordX = 36;

current.sex = checkNumeral(coordX, coordY, current.sex, 1);

break;

}

case 4: {

coordX = 38;

rewriteString(coordX, coordY, PROF\_LENGTH, current.prof);

break;

}

case 5: {

coordX = 50;

current.exp = checkNumeral(coordX, coordY, current.exp, 2);

break;

}

case 6: {

coordX = 55;

do{

if((current.rank < 1) || (current.rank > 3)) {

SetColor(4,0);

}

current.rank = checkNumeral(coordX, coordY, current.rank, 1);

SetColor(7,0);

} while ((current.rank < 1) || (current.rank > 3));

break;

}

case 7: {

coordX = 60;

current.factoryNumber = checkNumeral(coordX, coordY, current.factoryNumber, 2);

break;

}

case 8: {

coordX = 66;

do{

if((current.deportmentNumber < 1) || (current.deportmentNumber > 10)) {

SetColor(4,0);

}

current.deportmentNumber = checkNumeral(coordX, coordY, current.deportmentNumber, 2);

SetColor(7,0);

} while ((current.deportmentNumber < 1) || (current.deportmentNumber > 10));

break;

}

case 9: {

coordX = 69;

current.salary = (float) checkNumeral(coordX, coordY, (long int) current.salary, 7);

break;

}

}

}

case 77: {

if (currentField >= countOfFields - 1) currentField = 0;

else currentField++;

break;

}

case 75: {

if (currentField <= 0) currentField = countOfFields - 1;

else currentField--;

break;

}

case 96:

case 241: {

if (menuInterface(acceptMessage, acceptMessageItemsCount)) {

return current;

}

break;

}

case 27: {

return mainData;

} } }}

/\*

\* ФУНКЦИЯ УДАЛЕНИЯ СПИСКА

\*/

int deleteList(list \*&top) {

if (top != NULL) {

list \*temp;

int i = 0;

for (temp = top; top != NULL; temp = top) {

top = temp->next;

delete temp;

}

if ((top == NULL) && (temp == NULL)) return 1;

else return 0;

}}

/\*

\* ФУНКЦИЯ УДАЛЕНИЯ ОДНОГО ЭЛЕМЕНТА СПИСКА

\*/

int deletePersonalData(list \*&listHead, list \*&listEnd, list \*current) {

if ((current == listHead) && (current == listEnd)) //удаление единственного элемента

{

listHead = NULL;

listEnd = NULL;

delete current;

return 0;

} else if (current == listHead) { //из начала списка

listHead = listHead->next;

listHead->pred = NULL;

delete current;

return 1;

} else if (current == listEnd) {//с конца

listEnd = listEnd->pred;

listEnd->next = NULL;

delete current;

return 2;

} else { //удаление из середины

current->pred->next = current->next;

current->next->pred = current->pred;

delete current;

return 3;

}

}

/\*

\* ФУНКЦИЯ ПРОСМОТРА ОЧЕРЕДИ С ВОЗМОЖНОСТЬЮ НАВИГАЦИИ(ПО ЗАПИСЯМ, А ТАК ЖЕ ПО СТРАВНИЦАМ)

\* mode:

\* 0 - функция для удаления элементов

\* 1 - для сохранения выведенного списка в файл

\*/

void viewList(list \*&listHead, list \*&listEnd, unsigned mode) {

if (listHead != NULL) {

list \*currentL, \*temp, \*startDisplay;

int i, currentNum = 1;

startDisplay = currentL = listHead;

short int delResult;

while (listHead != NULL) {

system("cls");

temp = startDisplay;

drawTableHead(sort\_mode);

for (i = 1; (i <= countOfDisplayRecords) && (temp != NULL); temp = temp->next, i++) {

if (i == currentNum) {

SetColor(0, 8);

}

outData(temp);

SetColor(7, 0);

};

drawHelpMenu(mode);

switch (key = getch()) {

case 72: {

if (currentL->pred != NULL) {

--currentNum;

currentL = currentL->pred;

}

break;

}

case 80: {

if (currentL->next != NULL) {

++currentNum;

currentL = currentL->next;

}

break;

}

case 75: {

currentNum = 1;

temp = startDisplay;

for (i = 1; (i++ <= countOfDisplayRecords) && (temp != NULL); temp = temp->pred);

if (temp == NULL) {

temp = startDisplay;

}

startDisplay = currentL = temp;

break;

}

case 77: {

currentNum = 1;

temp = startDisplay;

for (i = 1; (i++ <= countOfDisplayRecords) && (temp != NULL); temp = temp->next);

if (temp == NULL) {

temp = startDisplay;

}

startDisplay = currentL = temp;

break;

}

case 83: {

if (!mode) {

SetColor(12, 0);

printf("ХОТИТЕ ЛИ ВЫ УДАЛИТЬ ЭТУ ЗАПИСЬ? \n");

SetColor(7, 0);

if (menuInterface(acceptMessage, acceptMessageItemsCount) == 1) {

if (currentL == listHead) {

delResult = deletePersonalData(listHead, listEnd, currentL);

currentL = startDisplay = listHead;

currentNum = 1;

} else {

if (currentL == startDisplay) {

temp = startDisplay;

for (i = 1; (i++ <= countOfDisplayRecords) && (temp != NULL); temp = temp->pred);

startDisplay = temp;

}

currentL = currentL->pred;

delResult = deletePersonalData(listHead, listEnd, currentL->next);

currentNum--;

} } }

break; }

case 96: {

if (mode) {

cleanStatusBar();

printf("ВВЕДИТЕ ИМЯ ФАЙЛА: ");

char file\_name[MAX\_STR\_LENGTH];

char default\_name[MAX\_STR\_LENGTH + 11] = "selection\_";

cin.getline(file\_name, MAX\_STR\_LENGTH);

if (cin.fail()) { //ПРИ ПЕРЕПОЛНЕНИИ БУФЕРА ВХОДНОГО ПОТОКА

cin.clear(); //СБРОС ОШИБКИ ПОТОКА

cin.ignore(1000, '\n'); //ИГНОРИРОВАНИЕ ОСТАВШИХСЯ В ПОТОКЕ СИМВОЛОВ

}

strcat(default\_name, file\_name);

if (!saveFile(listHead, default\_name)) {

printf("ВЫБОРКА СОХРАНЕНА");

} else {

printf("ВЫБОРКА НЕ СОХРАНЕНА!");

}

getch();

}

break;

}

case 13: {

if (!mode) {

currentL->inf = editData(currentL->inf);

}

break;

}

case 27: { return; } }

if (currentNum > countOfDisplayRecords) {

currentNum = 1;

startDisplay = currentL;

} else if ((currentNum < 1)) {

currentNum = countOfDisplayRecords;

temp = startDisplay;

for (i = 1; (i <= countOfDisplayRecords) && (temp != NULL); temp = temp->pred, i++);

if (temp == NULL) {

temp = startDisplay;

}

startDisplay = temp;

}

}

cleanStatusBar();

emptyMessage();

} else return;

}

-------------------------------------------------------------------------------------------------------

//Файл “CP\_sort.cpp”

#include "CP\_types.h"

/\*

\* ФУНКЦИЯ СОРТИРОКИ СПИСКА ПО КРИТЕРИЯМ

\* mode:

\* 0 - по фамилии

\* 1 - по индивидуальному номеру

\* 2 - по проффессии и фамилии

\* 3 - по номеру цеха(factory) и фамилии

\* 4 - по номеру участка(deportment) и фамилии

\* 5 - по заработной плате(salary) и фамилии

\* 6 - по полу(sex) и фамилии

\* 7 - по проффессии и разряду

\*/

int sort(list \*&head, list \*&end, short int mode) {

if (head != NULL) {

switch (mode) {

case 0: {

list \*temp1, \*temp2;

tableData tempData;

temp1 = temp2 = NULL;

for (temp1 = head; temp1->next != NULL; temp1 = temp1->next) {

for (temp2 = temp1->next; temp2 != NULL; temp2 = temp2->next) {

if (strcmp(temp1->inf.fio, temp2->inf.fio) > 0) {

tempData = temp1->inf;

temp1->inf = temp2->inf;

temp2->inf = tempData;

}

}

}

return mode;

}

case 1: {

list \*temp1, \*temp2;

tableData tempData;

temp1 = temp2 = NULL;

for (temp1 = head; temp1->next != NULL; temp1 = temp1->next) {

for (temp2 = temp1->next; temp2 != NULL; temp2 = temp2->next) {

if (temp1->inf.personalNumber > temp2->inf.personalNumber) {

tempData = temp1->inf;

temp1->inf = temp2->inf;

temp2->inf = tempData;

} } }

return mode;

}

case 2: {

list \*temp1, \*temp2;

tableData tempData;

temp1 = temp2 = NULL;

for (temp1 = head; temp1->next != NULL; temp1 = temp1->next) {

for (temp2 = temp1->next; temp2 != NULL; temp2 = temp2->next) {

if (strcmp(temp1->inf.prof, temp2->inf.prof) > 0) {

tempData = temp1->inf;

temp1->inf = temp2->inf;

temp2->inf = tempData;

} else if (strcmp(temp1->inf.prof, temp2->inf.prof) == 0) {

if (strcmp(temp1->inf.fio, temp2->inf.fio) > 0) {

tempData = temp1->inf;

temp1->inf = temp2->inf;

temp2->inf = tempData;

} } } }

return mode;

}

case 3: {

list \*temp1, \*temp2;

tableData tempData;

temp1 = temp2 = NULL;

for (temp1 = head; temp1->next != NULL; temp1 = temp1->next) {

for (temp2 = temp1->next; temp2 != NULL; temp2 = temp2->next) {

if (temp1->inf.factoryNumber > temp2->inf.factoryNumber) {

tempData = temp1->inf;

temp1->inf = temp2->inf;

temp2->inf = tempData;

} else if (temp1->inf.factoryNumber == temp2->inf.factoryNumber) {

if (strcmp(temp1->inf.fio, temp2->inf.fio) > 0) {

tempData = temp1->inf;

temp1->inf = temp2->inf;

temp2->inf = tempData;

} } } }

return mode;

}

case 4: {

list \*temp1, \*temp2;

tableData tempData;

temp1 = temp2 = NULL;

for (temp1 = head; temp1->next != NULL; temp1 = temp1->next) {

for (temp2 = temp1->next; temp2 != NULL; temp2 = temp2->next) {

if (temp1->inf.deportmentNumber > temp2->inf.deportmentNumber) {

tempData = temp1->inf;

temp1->inf = temp2->inf;

temp2->inf = tempData;

} else if (temp1->inf.deportmentNumber == temp2->inf.deportmentNumber) {

if (strcmp(temp1->inf.fio, temp2->inf.fio) > 0) {

tempData = temp1->inf;

temp1->inf = temp2->inf;

temp2->inf = tempData;

} } } }

return mode;

}

case 5: {

list \*temp1, \*temp2;

tableData tempData;

temp1 = temp2 = NULL;

for (temp1 = head; temp1->next != NULL; temp1 = temp1->next) {

for (temp2 = temp1->next; temp2 != NULL; temp2 = temp2->next) {

if (temp1->inf.salary > temp2->inf.salary) {

tempData = temp1->inf;

temp1->inf = temp2->inf;

temp2->inf = tempData;

} else if (temp1->inf.salary == temp2->inf.salary) {

if (strcmp(temp1->inf.fio, temp2->inf.fio) > 0) {

tempData = temp1->inf;

temp1->inf = temp2->inf;

temp2->inf = tempData;

} } } }

return mode;

}

case 6: {

list \*temp1, \*temp2;

tableData tempData;

temp1 = temp2 = NULL;

for (temp1 = head; temp1->next != NULL; temp1 = temp1->next) {

for (temp2 = temp1->next; temp2 != NULL; temp2 = temp2->next) {

if (temp1->inf.sex > temp2->inf.sex) {

tempData = temp1->inf;

temp1->inf = temp2->inf;

temp2->inf = tempData;

} else if (temp1->inf.sex == temp2->inf.sex) {

if (strcmp(temp1->inf.fio, temp2->inf.fio) > 0) {

tempData = temp1->inf;

temp1->inf = temp2->inf;

temp2->inf = tempData;

} } } }

return mode;

}

case 7: {

list \*temp1, \*temp2;

tableData tempData;

temp1 = temp2 = NULL;

for (temp1 = head; temp1->next != NULL; temp1 = temp1->next) {

for (temp2 = temp1->next; temp2 != NULL; temp2 = temp2->next) {

if (strcmp(temp1->inf.prof, temp2->inf.prof) > 0) {

tempData = temp1->inf;

temp1->inf = temp2->inf;

temp2->inf = tempData;

} else if (strcmp(temp1->inf.prof, temp2->inf.prof) == 0) {

if (temp1->inf.rank > temp2->inf.rank) {

tempData = temp1->inf;

temp1->inf = temp2->inf;

temp2->inf = tempData;

} } } }

return mode;

}

case 27: {

return -2;

}

}

} else {

cleanStatusBar();

emptyMessage();

return -1;

}

}

/\*

\* ФУНКЦИЯ ДЛЯ ПОИСКА ДАННЫХ ПО КЛЮЧЕВОМУ ПОЛЮ

\* ВОЗВРАЩАЕТ 1 - В СЛУЧАЕ ПУСТОГО СПИCКА

\*/

int searchData(list \*&head, short int mode) {

if (head == NULL) {

emptyMessage();

return 1;

}

list \*temp, \*tempHead, \*tempEnd;

tempHead = tempEnd = NULL;

bool searchResult = false;

cleanStatusBar();

switch (mode) {

case 27: {

return 27;

}

case 0: {

printf("ВВЕДИТЕ КЛЮЧ ПОИСКА: ");

char FIO[FIO\_LENGTH];

cin.getline(FIO, FIO\_LENGTH);

if (cin.fail()) {

cin.clear();

cin.ignore(1000, '\n');

}

strToFormat(FIO, FIO\_LENGTH);

for (temp = head; temp != NULL; temp = temp->next) {

if (strstr(temp->inf.fio, FIO)) {

searchResult = true;

break;

//outData(temp);

}

}

if (searchResult) {

system("cls");

drawTableHead();

temp->inf = editData(temp->inf);

}

break;

}

case 1: {

printf("ВВЕДИТЕ КЛЮЧ ПОИСКА: ");

long int tempNumber = checkNumeral(21, 19, 0, 6);

for (temp = head; temp != NULL; temp = temp->next) {

if (temp->inf.personalNumber == tempNumber) {

searchResult = true;

break;

}

}

if (searchResult) {

system("cls");

drawTableHead();

temp->inf = editData(temp->inf);

}

break;

}

case 2: {

printf("ВВЕДИТЕ КЛЮЧ ПОИСКА: ");

long int tempNumber = checkNumeral(21, 19, 0, 2);

for (temp = head; temp != NULL; temp = temp->next) {

if (temp->inf.rank == tempNumber) {

searchResult = true;

if (tempHead == NULL) {

organizeList(tempHead, tempEnd, temp->inf);

} else {

addPerson(tempEnd, temp->inf);

}

}

}

viewList(tempHead, tempEnd, 1);

break;

}

case 3: {

printf("ВВЕДИТЕ КЛЮЧ ПОИСКА: ");

long int tempNumber = checkNumeral(21, 19, 0, 2);

for (temp = head; temp != NULL; temp = temp->next) {

if (temp->inf.factoryNumber == tempNumber) {

searchResult = true;

if (tempHead == NULL) {

organizeList(tempHead, tempEnd, temp->inf);

} else {

addPerson(tempEnd, temp->inf);

}

}

}

viewList(tempHead, tempEnd, 1);

break;

}

case 4: {

printf("ВВЕДИТЕ КЛЮЧ ПОИСКА: ");

long int tempNumber = checkNumeral(21, 19, 0, 2);

for (temp = head; temp != NULL; temp = temp->next) {

if (temp->inf.deportmentNumber == tempNumber) {

searchResult = true;

if (tempHead == NULL) {

organizeList(tempHead, tempEnd, temp->inf);

} else {

addPerson(tempEnd, temp->inf);

} } }

viewList(tempHead, tempEnd, 1);

break; } }

if (!searchResult) { cout << "ДАННЫЕ НЕ НАЙДЕНЫ" << endl;

}

deleteList(tempHead); return 0;}

/\*

\* ФУНКЦИЯ ПОДСЧЁТА СРЕДНЕЙ ЗАРПЛАТЫ И КОЛИЧЕСТВА РАБОТНИКОВ НА УЧАСТКЕ

\* СОХРАНЯЕТ ДАННЫЕ В ФАЙЛ workers\_register.txt

\* ВЫВОДИТ СОДЕРЖИМОЕ ФАЙЛА НА ЭКРАН

\*/

void registerOfWorkers(list \*head) {

struct workers\_register {

double total\_salary = 0, everage\_salary = 0;

unsigned int rank[3] = {0, 0, 0}, total\_count\_workers = 0;

} deportments[deportment\_count];

list \*temp;

unsigned depTempNum, rankTempNum;

for (temp = head; temp != NULL; temp = temp->next) {

depTempNum = temp->inf.deportmentNumber - 1;

rankTempNum = temp->inf.rank - 1;

deportments[depTempNum].total\_count\_workers++;

deportments[depTempNum].total\_salary += temp->inf.salary;

deportments[depTempNum].rank[rankTempNum]++;

}

system("cls");

char out\_file\_name[MAX\_STR\_LENGTH] = "workers\_register.txt";

/\*printf("ENTER FILE NAME: ");

cin.getline(out\_file\_name, MAX\_STR\_LENGTH);

if (cin.fail()) { //ПРИ ПЕРЕПОЛНЕНИИ БУФЕРА ВХОДНОГО ПОТОКА

cin.clear(); //СБРОС ОШИБКИ ПОТОКА

cin.ignore(1000, '\n'); //ИГНОРИРОВАНИЕ ОСТАВШИХСЯ В ПОТОКЕ СИМВОЛОВ

}\*/

FILE \*out\_file = fopen(out\_file\_name, "wt");

if (!out\_file) {

printf("ФАЙЛ НЕ СОХРАНЕН");

} else {

for (int i = 0; i < console\_row\_length - 27; i++) {

fprintf(out\_file, "-");

}

fprintf(out\_file, "\n\t|\t\t | EMPLOYEES | | ");

fprintf(out\_file, "\n DEP # | EVERAGE SALARY |-----------------| TOTAL |\n");

fprintf(out\_file, "\t|\t\t | 1 | 2 | 3 | |\n");

for (int i = 0; i < console\_row\_length - 27; i++) {

fprintf(out\_file, "-");

}

fprintf(out\_file, "\n");

for (int i = 0; i < deportment\_count; i++) {

if (deportments[i].total\_salary) {

deportments[i].everage\_salary = deportments[i].total\_salary / deportments[i].total\_count\_workers;

}

fprintf(out\_file, "%7d | %14.2f | %3d | %3d | %3d | %4d |\n", i + 1, deportments[i].everage\_salary,

deportments[i].rank[0], deportments[i].rank[1], deportments[i].rank[2],

deportments[i].total\_count\_workers);

}

for (int i = 0; i < console\_row\_length - 27; i++) {

fprintf(out\_file, "-");

}

fprintf(out\_file,"\n");

fclose(out\_file);

system("copy workers\_register.txt con");

printf("ВЕДОМОСТЬ СОХРАНЕНА В ФАЙЛ: workers\_register.txt");

return;

}}