Министерство образования и науки Российской Федерации

Федеральное государственное образовательное учреждение

высшего образования

«Севастопольский государственный университет»

Кафедра информационных систем

Курсовой проект по дисциплине

«Основы программирования и алгоритмические языки»

на тему:

«Программа для подсчета количества сотрудников на предприятии»

Пояснительная записка

СГУ 2015.13

Листов 62

Студента 2 курса группы ИС-22-о

направление подготовки 09.03.02

(подпись)\_\_\_\_\_\_\_\_\_ Серегин А.В.

« » \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2015г.

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(должность, учебное звание, фамилия и инициалы)

Оценка:\_\_\_\_\_\_\_\_\_\_\_\_\_

Члены комисии \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (фамилия и инициалы)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (фамилия и инициалы)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (фамилия и инициалы)

г. Севастополь

2015 г.

АННОТАЦИЯ

В документе описывается кроссплатформенное приложение, написанное на языке С/С++ с использованием библиотеки Qt, для реализации программного интерфейса. Программа предназначена для подсчета как общего количества сотрудников, мужчин и женщин, работающих на предприятии, а также их процентная зависимость от общего кол-ва работников всех предприятий.

Программа разработана в соответствии с постановкой задачи на курсовое проектирование по дисциплине «Основы Программирования и Алгоритмические Языки» на тему «Обработка данных, представленных в форме таблицы».

Документ содержит технические подробности реализации и описание возможностей программы.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ………………………………………………………………………..4

1. НАЗНАЧЕНИЕ И ОБЛАСТЬ ПРИМЕНЕНИЯ……………………………….5

2. ТЕХНИЧЕСКИЕ ХАРАКТЕРИСТИКИ………………………………………6

2.1. Постановка задачи……………………………………………………..8

2.2. Описание основных типов данных, используемых в программе…..8

2.3. Стуктура программы…………………………………………………11

2.4. Описание файлов и их данных………………………………………11

2.4.1. main.cpp………………………………………………………11

2.4.2. mainwindow.h………………………………………………...11

2.4.3. Data.h…………………………………………………………12

2.4.4. WorkersData.h………………………………………………..13

2.4.5. Vector.h……………………………………………………….14

2.4.6. Worker.h………………………………………………………15

2.4.7. FactoryesData.h……………………………………………….15

2.4.8. Tree.h………………………………………………………….16

2.4.9. macro.h………………………………………………………..16

2.4.10. searchdialog.h………………………………………………..16

2.4.11. workerdialog.h……………………………………………….16

2.5. Структурные схемы функций………………………………………..17

3. ВЫПОЛНЕНИЕ ПРОГРАММЫ……………………………………………..21

3.1. Условие выполнения программы……………………………………21

3.2. Загрузка, запуск, проверка работоспособности программы……….21

ВЫВОДЫ………………………………………………………………………...29

ПРИЛОЖЕНИЕ А……………………………………………………………….30

ВВЕДЕНИЕ

Целью данного курсового проекта является разработка программы подсчета количества сотрудников предприятия, исходя из общей таблицы всех сотрудников.

В документе описывается программа наименованием «Workers Table», которая разрабатывалась по заданию кафедры Информационных систем Севастопольского Государственного Университета на основании задания варианта №13 из методических указаний с 2015г.

В настоящее время для эффективной работы любого предприятия используются автоматизированные системы, значительно упрощающие работу сотрудников. Созданная программа позволяет быстро организовать список сотрудников всех предприятий, а также корректировать данные о каждом из них.

1. НАЗНАЧЕНИЕ И ОБЛАСТЬ ПРИМЕНЕНИЯ

Назначение программы «Workers Table» состоит в хранении информации о количестве сотрудников на каждом предприятии.

Данные представлены в виде таблицы, содержащей:

* идентификационный номер предприятия;
* общее количество сотрудников;
* количество мужчин;
* количество женщин;
* % сотрудников от общего количества сотрудников всех предприятий;
* % мужчин от общего количества мужчин всех предприятий;
* % женщин от общего количества женщин всех предприятий.

Также можно просмотреть таблицу всех сотрудников, содержащую:

* идентификационный номер;
* ФИО;
* дата рлождения;
* пол;
* профессию;
* опыт работы;
* разряд;
* идентификационный номер предприятия;
* номер участка, на котором работает сотрудник;
* заработная плата.

Программа может использоваться на предприятии, предназначенном для разделения труда.

2. ТЕХНИЧЕСКИЕ ХАРАКТЕРИСТИКИ

2.1. Постановка задачи

Согласно заданию, требуется написать программу подсчета сотрудников на предприятиях. Программа должна осуществлять ввод и вывод данных из/в двоичный файл, а также сохранять таблицу предприятий в текстовом формате.

По заданию в варианте дана входная таблица сотрудников и выходная таблица предприятий. Выходная таблица должна формироваться по данным, хранящимся во входной таблице.

В программе требуется реализовать:

- просмотр таблиц сотрудников и предприятий;

- добавление нового сотрудника;

- редактирование данных о сотруднике;

- поиск сотрудника по ID;

- удаление сотрудников из списка;

- считывание и сохранение данных в двоичный файл;

- вывод выходной таблицы в текстовый файл;

- возможность сортировки данных по полям, возрастанию и убыванию.

Входная таблица содержит:

* идентификационный номер;
* ФИО;
* дата рлождения;
* пол;
* профессию;
* опыт работы;
* разряд;
* идентификационный номер предприятия;
* номер участка, на котором работает сотрудник;
* заработная плата.

Таблица 2.1.1. – Таблица выходных данных.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № цеха | Кол-во рабочих | | | Кол-во рабочик, % | | |
| Муж. | Жег. | Всего | Муж. | Жен. | Всего |
|  |  |  |  |  |  |  |

2.2. Описание основных типов данных, используемых в программе

Структура хранения всех данных:

struct Data

{

worker::WD workersData; //список сотрудников

factory::FD factoryesData; //список предприятий

size\_t all; //количество рабочих

size\_t male; //количество женщин

size\_t female; //количество мужчин

};

Структура хранения сотрудников:

struct WorkersData

{

SP sortPriority; //приоритет сортировки сотрудников

Worker\* workers; //вектор сотрудников

size\_t size; //кол-во сотрудников

size\_t capacity; //занимаемая вектором память/sizeof(worker)

};

Данные о сотрудниках хранятся в векторе(динамическом массиве):

Сравнения вектора с другими структурами:

* выполнение сортировки за O(n log n), сортировать связный список возможно только за O(n2), что в разы медленнее;
* доступ к любому из элементов за O(1), в списке O(n), дерево O(log n);
* удаление за О(n), в списке имея указатель на элемент O(1), не имея O(n), удаление из дерева так же за линейное время;
* вставка элемента в вектор и список осуществляется за линейное время, тут выигрывает дерево O(log n).

По приведенным выше результатам был выбран вектор, в связи с тем, что операции над ним проводятся быстрее чем над связным списком. В сравнении с деревом вектор примерно одинаковый по скорости работы, тем не менее дерево после удаления из него данных стремится к линейному списку, что начинает уменьшать его быстродействие, а для исправления этого нужно писать функции уравновешивания дерева по уровням, что не так-то и просто.

Структура хранения вектора:

struct WorkersData

{

SP sortPriority; //приоритет сортировки сотрудников

Worker\* workers; //вектор сотрудников

size\_t size; //кол-во сотрудников

size\_t capacity; //занимаемая вектором память/sizeof(worker)

};

Структура записи о сотруднике:

struct Worker

{

size\_t id; //id сотрудника

::FIO fio; //ФИО

Date birthday; //дата рождения

Gender gender; //пол

char profession[20];//профессия

size\_t experience; //опыт работы

size\_t rang; //ранг

size\_t factory; //id предприятия

size\_t sector; //номер сектора

double payment; // З/П

};

Структура хранения записей о предприятиях:

struct FactoryData

{

SP sortPriority; //приоритет сортировки

FNode\* root; //указатель на корень дерева

size\_t size; //кол-во эл-тов дерева

};

Для хранения данных о предприятиях было выбрано двоичное дерево, так как количество предприятий обычно значительно меньше количества сотрудников, удаление предприятий достаточно редкое явление, что не приводит к «деформации дерева». Так как в основном для списка предприятий используются только функции просмотра, и поиска предприятия, а поиск осуществляется за O(log n) против O(n) вектора, было выбрано дерево.

Структура узла бинарного дерева:

struct FNode

{

Factory\* key;

FNode\* left;

FNode\* right;

};

Структуры записи о предприятии:

//количество сотрудников на предприятии

struct CountWorkers

{

size\_t all; //количесво всех сотрудников

size\_t male; //кол-во мужчин

size\_t female; //кол-во женщин

};

//процент работников на предприятии от общего

struct PersentCountWorkers

{

size\_t all;

size\_t male;

size\_t female;

};

struct Factory

{

size\_t number; //номер предприятия

CountWorkers count;

PersentCountWorkers persentCount;

};

2.3. Структура программы

На рисунке ниже представлена схема включения файлов программы «Workers Table».

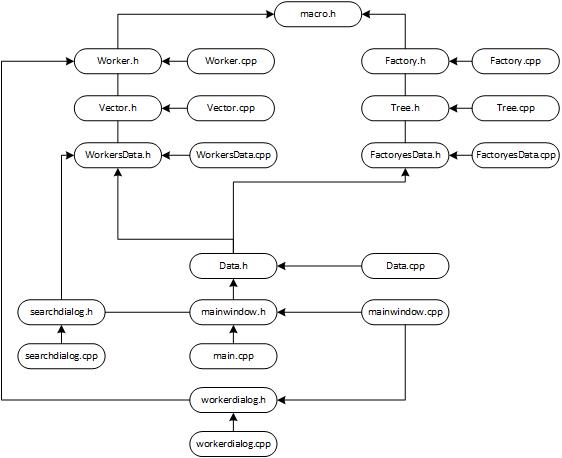


Рисунок 2.1. – Схема включения файлов программы.

2.4. Описание файлов и их данных

2.4.1. main.cpp

Файл хранит в себе функцию main(), которая создает диалоговое окно, устанавливает его основные параметры и возвращает значение, которое возвращается обработчиком событий данного диалогового окна по его завершению.

2.4.2. mainwindow.h

Заголовочный файл хранящий класс основного окна class MainWindow : public QMainWindow, в котором хранятся все данные о предприятиях и сотрудниках, а так же о всех его элементах взаимодействия с пользователем.

2.4.3. Data.h

Заголовочный файл, содержащий структуру, хранящую данные о всех сотрудниках и предприятиях, а также прототипы функций, обрабатывающих эту струтуру.

Список функций:

* void addWorker(Data& \_data, const worker::Worker& \_newWorker);

Функция addWorker добавляет нового сотрудника в вектор, а так же пересчитывает в выходной таблице все количественные данные, в случае отсутствия предприятия добавляется новое с первым сотрудником(данным).

* void delWorker(Data& \_data, const size\_t& \_dataPos);

Функция delWorker удаляет сотрудника из вектора, по его позиции в нем, пересчитывает в выходной таблице все количественные данные, в случае удаления последнего сотрудника на данном предприятии удаляется и само предприятие.

* void saveBinDataFile(const Data& \_data, const std::string& \_path);

Функция saveBinDataFile сохраняет количество сотрудников и список сотрудника в двоичный файл.

* void loadBinDataFile(Data& \_data, const std::string& \_path);

Функция loadBinDataFile освобождает память от прошлых данных, загружая новые из двоичного файла, считывает поэлементно (по одному сотруднику) и добавляет его в новую «базу».

- void printFactoryTableTextFile(const Data& \_data, const std::string& \_path);

Функция printFactoryTableTextFile выводит выходную таблицу в текстовый файл.

- void clearData(Data& \_data);

Функция clearData освобождает память от данных.

- void initialize(Data& \_data);

Функция clearData инициализирует данные начальными значеними.

2.4.4. WorkersData.h

Заголовочный файл, содержащий структуру, хранящую данные о всех сотрудниках, а также прототипы функций, обрабатывающих эту струтуру.

Список функций:

- void add(WD& workers, const Worker& \_newWorker);

Функция add добавляет нового работника в общий список.

- long long search(const WD& workers, const size\_t& \_id);

Функция search ищет сотрудника в списке и возвращает его позицию в нем или -1 в случае неудачного поиска.

- void del(WD& workers, const size\_t& \_dataPos);

Функция del удаляет сотрудника из списка по его позиции в ней.

- void sort(WD& workers, int sortFlag = 0);

Функция sort сортирует список по возрастанию если sortFlag = 0 и пр убыванию при значении 1.

- void clear(WD& workers);

Функция clear освобождает память от вектора сотрудников.

2.4.5. Vector.h

Заголовочный файл хранящий набор функций для управления вектором, хранящим записи о работниках.

Список функций:

* void resize(Worker\*\* vector, const size\_t& \_size,

const size\_t& \_newSize, size\_t& capacity);

Функция resize изменяет размер вектора, создавая в памяти вектор нового рамера и переписывая данные из исходного в случае если newSize > capacity.

* void erase(Worker\*\* vector, size\_t& size,

const size\_t& \_first, const size\_t& \_last);

Функция erase удаляет элементы вектора на заданных границах, сдвигая правые элементы на их места.

* void insert(Worker\*\* vector, size\_t& size,

size\_t& capacity, const Worker& \_new, const size\_t& \_pos);

Функция insert вставляет новый элемент на заданную позицию, сдвигая все элементы от этой позиции на единицу вправо.

* void quickSort(Worker\*\* arr, const int& \_left,

const int& \_right, SP \_priority, CMPFUNC(cmpfunc));

Функция quickSort сортирует вектор согласно приоритету сортировки, по известному алгоритму быстрой сортировки.

2.4.6. Worker.h

Заголовочный файл описывающий запись о сотруднике, а также хранящий прототип функции сортировки.

int compareWorker(const Worker& \_left, const Worker& \_right, SP sortPriority);

Функция compareWorker сравнивает левое и правое значение по заданному полю, возвращает -1 при старшенстве первого, 1 – второго и 0 при их равенстве.

2.4.7. FactoryesData.h

Заголовочный файл хранящий структуру, содержащую данные о всех предприятиях, а так же набор функций по ее обработке.

Список функций:

* void add(FD& factoryes, Factory\* \_factory);

Функция add добавляет новое предприятие в список всех предприятий.

* Factory\* search(const FD& \_factoryes, const size\_t& \_id);

Функция search ищет в списке указатель на предприятие соответствующее заданному ID или возвращает nulltpr в результате его отсутсвия.

* void del(FD& factoryes, const size\_t& \_id);

Функция del удаляет предприятие по заданному id из списка.

* void sort(FD& factoryes, int cmpFlag = 0);

Функция sort сортирует список данных, по направлению согласно флагу сортировки 0 – убывания, 1 – возрастания.

* void clearData(FD& factoryes);

Функция clearData освобождает память от данных.

2.4.8. Tree.h

Заголовочный файл, хранящий структуру узла двоичного дерева, а также набор функций способствующих построению самого дерева.

Список функций:

* void add(FNode\*\* root, Factory\* \_key, SP \_priority, CMPFUNC(cmpfunc));

Функция add добавляет новое предприятие в дерево согласно установленному приоритету.

* Factory\* search(FNode\* \_root, const size\_t& \_number);

Функция search ищет в списке указатель на предприятие соответствующее заданному ID или возвращает nulltpr в результате его отсутсвия.

* void del(FNode\*\* root, const size\_t& \_number);

Функция del удаляет предприятие по заданному id из списка.

* void reBuild(FNode\*\* root, SP \_priority, CMPFUNC(cmpfunc));

Функция reBuild перестранивает исходное дерево в новое, согласно приоритету.

* void clear(FNode\*\* root);

Функция clear освобождает память от дерева.

2.4.9. macro.h

Заголовочный файл содержащий набор простых макросов.

2.4.10. searchdialog.h

Заголовочный файл содержащий класс диалогового окна, позволяющее пользователю задать ID сотрудника для поиска его данных.

2.4.11. workerdialog.h

Заголовочный файл содержащий класс диалогового окна, позволяющего заполнять или редактировать данные о сотруднике.

2.5. Структурные схемы функций

На рисунках ниже представлены структурные схемы алгоритмов функций.



Рисунок 2.2. – Структурная схема алгоритма

перестраивания дерева.



Рисунок 2.3. – Структурная схема алгоритма вызывающего

перестаивание дерева.



Рисунок 2.4. – Схема инициализации данных.



Рисунок 2.5. – Алгоритм очистки дерева.



Рисунок 2.6. – Алгоритм освобождения памяти

от данных.



Рисунок 2.7. –Алгоритм удаления данного узла дерева.



Рисунок 2.8. – Алгоритм удаление части вектора.



Рисунок 2.9. – Алгоритм вставки элемета

в вектор.



Рисунок 2.10. – Алгоритм изменения размера

вектора.

3. ВЫПОЛНЕНИЕ ПРОГРАММЫ

3.1. Условие выполнения программы

Данная программа предназначена для работы на ЭВМ под управлением различных операционных систем, так как приложение является кроссплатформенно компилируемым. Минимальные системные требования соответствуют минимуму требований операционной системы, так как программа легковесна, занимает минимум оперативной памяти и использует минимум ресурсов процессора.

Размер исполняемого файла составляет 131.1. кб.

Продукт разрабатывался и тестировался на компьютере конфигураций, приведенных ниже:

Процессор: Intel Core I5 – 4200M (2.5 GHz);

ОЗУ: 4Гб;

ОС: Gnu Linux Mint 17.2.

Для написания данного приложения был выбран язык Cи/С++, а также для реализации интерфейса была использована библиотека Qt.

3.2. Загрузка, запуск, проверка работоспособности программы

Для запуска программы требуется запустить исполняемый файл WorkersTable.

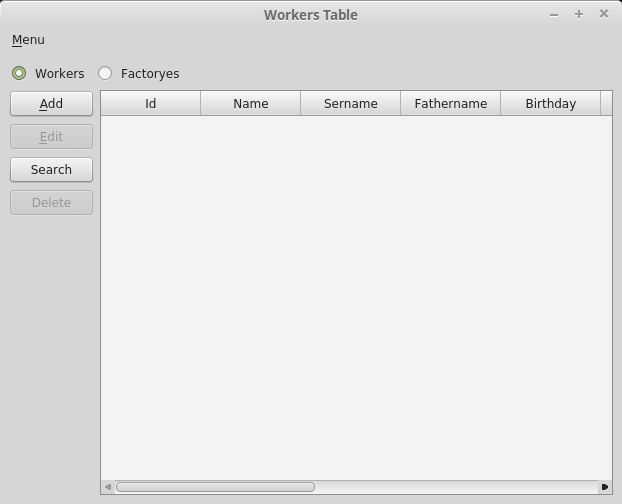


Рисунок 3.1. – Главное окно программы.

Запустив программу мы видим ее главное окно. В окне расположены элементы:

* Menu. Кликнув на него или нажав Alt+M появится список Menu:

Open – открыть бинарный файл.

Save – сохранить данные в новый бин. файл, или заменить существующий.

Save factoryes table .txt – сохранение таблици преприятий в файл.

Exit – закрытие программы.

* Управляемые кнопки Workers и Factoryes. Отвечают за отображаемую таблицу. Таблица Workers отображается по умолчанию.
* Кнопка Add. Вызывает диалоговое окно для добавления новых сотрудников. В нем присутствует набор полей для заполнения данных о сотрудниках, а также кнопка закрытия окна и добавления нового элемента. Добавлять элементы с одинаковыми ID запрещено, после добавления нового элемента окно не закрывается, но блокируется кнопка добавить. При изменении данных в полях кнопка тут же становится доступным.

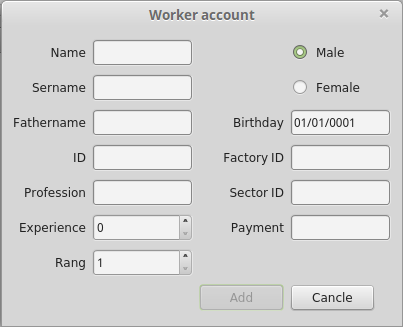


Рисунок 3.2. – Диалоговое окно добавления.

* Кнопка Edit. Доступна в случае выделения одного элемента в таблице Workres. По её нажатию открывается диалоговое окно с данными о выделенном работнике, такое же, как и окно добавления. Данное окно закроется после принятия изменений или нажатия кнопки отмены. В случае смены ID на уже существующий на экран отобразится ошибка.
* Кнопка Search. По её нажатии или нажатии сочетания клавиш Ctrl+F откроется диалоговое окно поиска сотрудника по его ID. Кнопка Search активна только тогда, когда в поле ID есть данные. По ее нажатии открывается окно редактирования сотрудника, в случае его находки или выдается соответсвующее сообщение об ошибке.

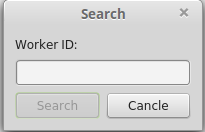


Рисунок 3.3. – Диалоговое окно поиска.

* Кнопка Delete. Активна когда в таблице Workers выделено один и более элементов. По ее нажатии данные элементы удаляются.

Добавим несколько элементов и посмотрим на результаты.

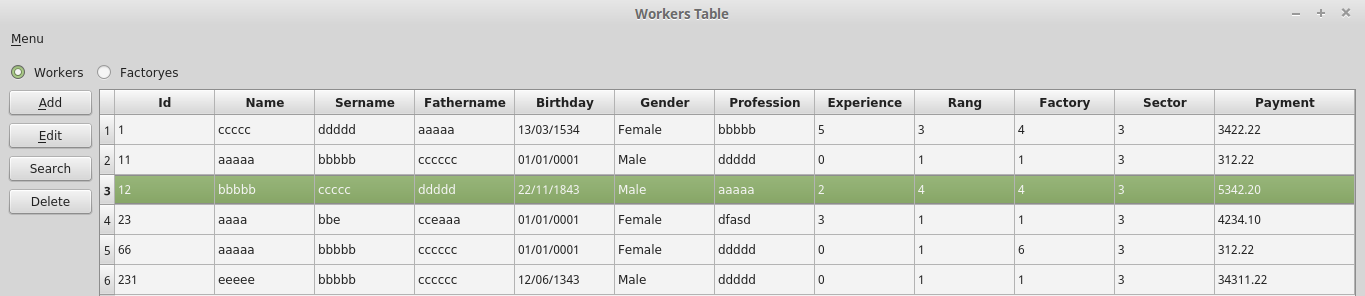


Рисунок 3.4. – Таблица работников после

добавления новых элементов.

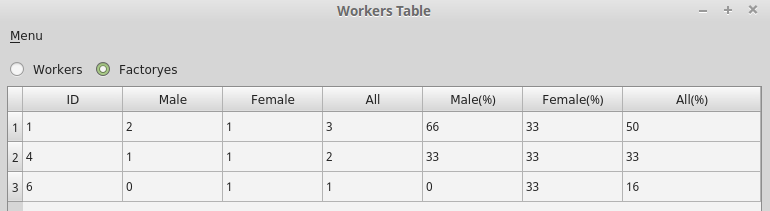


Рисунок 3.5. – Таблица предприятия после добавления

работников.

Сохраним данные в новый файл.

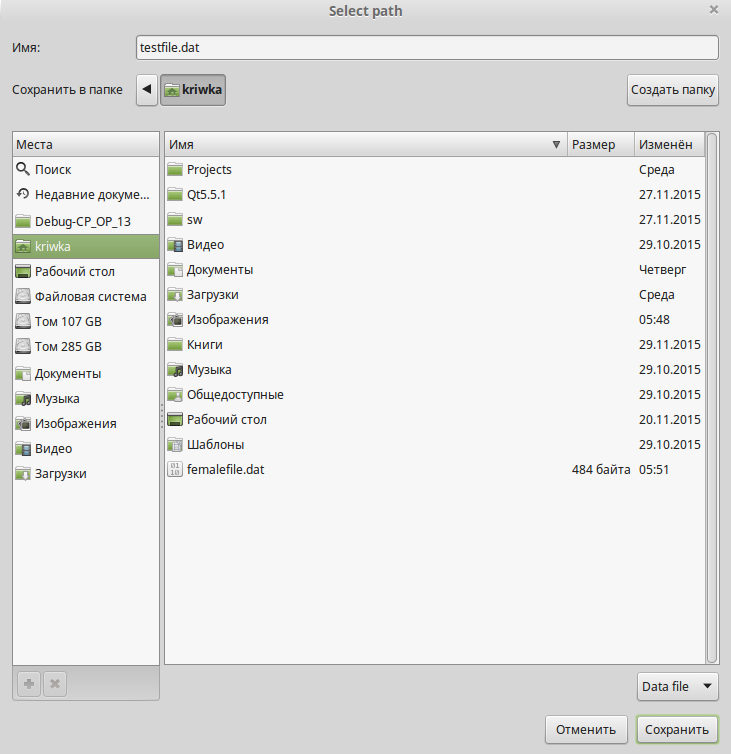


Рисунок 3.6. – Окно сохранения данных в файл.

Нажав на поле Name, в заголовке таблицы работников, отсортируем список в алфавитном порядке по именам.



Рисунок 3.7. – Таблица сотрудников

после сортировки.

Удалим всех мужчин и посмотрим на таблицы.

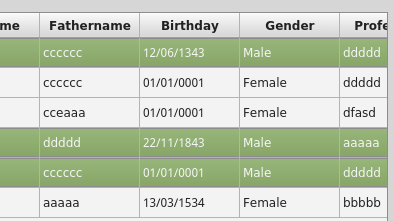


Рисунок 3.8. – Выделение мужчин.

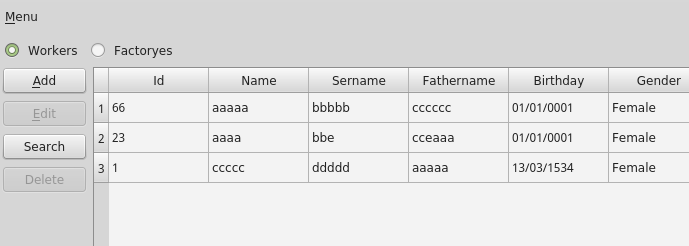


Рисунок 3.9. – Таблица работников после

удаления мужчин.

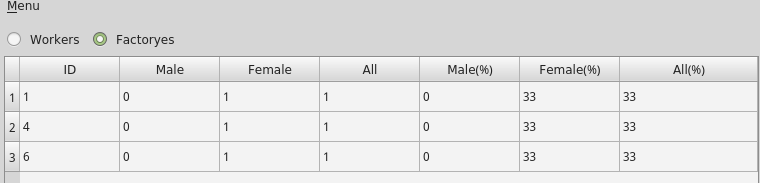


Рисунок 3.10. – Таблица предприятия после

удаления мужчин.

Загрузим первоначально сохраненный файл.

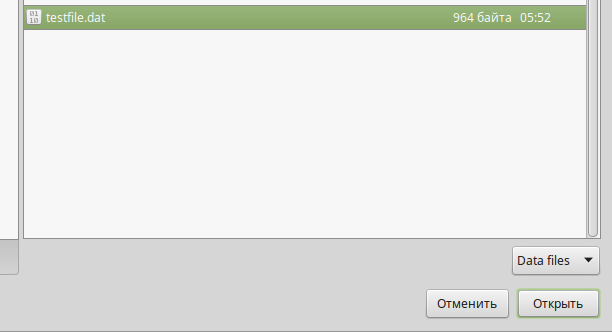


Рисунок 3.11. – Часть окна открытия файла.

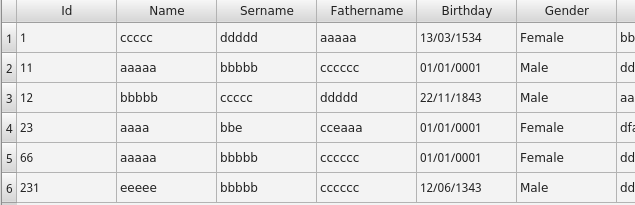


Рисунок 3.12. – Таблица работников после

загрузки файла.

Сохраним таблицу предприятий в текстовый файл.

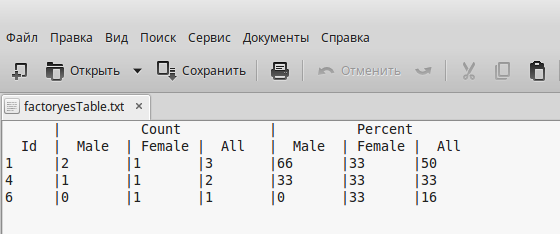


Рисунок 3.14. – Таблица в текстовом файле.

ВЫВОДЫ

Написанная программа была составлена в соответствии с постановкой задачи на курсовое проектирование по теме «обработка данных, представленных в виде таблицы» дисциплины «ОПиАЯ». В качестве структур хранения данных были выбраны вектор для таблицы сотрудников и двоичное дерево для выходной таблицы. Программа была реализована на языках Cи/Си++ с использованием библиотеки Qt для реализации пользовательского интерфейса, а также в связи с ее кроссплатформенностью, а так же легкостью создания графических интерфейса используя как просто классы библиотеки Qt, так и редактор форм. Основное окно было реализовано полностью программным кодом, а окно поиска и изменения данных работника построены на форме. Программа разрабатывалась в среде QtCreator.

Тестирование программы показали, что программа решает все задачи, поставленные в ТЗ. Маленькой недоработкой является отсутствие проверки вводимых данных в поле Birthday(можно ввести 32 день месяца или 15 месяц).

ПРИЛОЖЕНИЕ А

ТЕКСТ ПРОГРАММЫ

main.cpp

/\* Севастопольский Государственный Университет

\* Кафедра информационных систем

\* Программа WorkersTable

\* СГУ 2015.13

\* Разработал

\* ст. гр. ИС22-22-о

\* Серегин Александр Владимирович

\* 2015г.

\* Вариант задания №13 Утверждено 08.08.2015

\* Среда разработки QtCreator

\* Дата последней коррекции 20.10.2015

\* Версия 1.0

\*/

#include "mainwindow.h"

#include <QApplication>

int main(int argc, char \*argv[])

{

QApplication a(argc, argv);

MainWindow w;

w.setWindowTitle("Workers Table");

w.show();

return a.exec();

}

macro.h

#ifndef MACRO\_H

#define MACRO\_H

#define SP SortPriority

#define MIN(a, b) (((a) < (b)) ? (a) : (b))

#define MAX(a, b) (((a) > (b)) ? (a) : (b))

#define ABS(a) (((a) < 0) ? (-(a)) : (a))

typedef unsigned long long ULL;

typedef long long LL;

typedef unsigned int UI;

typedef unsigned long size\_t;

#define nullptr 0

#endif // MACRO\_H

Worker.h

#ifndef \_WORKER\_H\_

#define \_WORKER\_H\_

#include "macro.h"

#include <string>

//дата

struct Date

{

size\_t year;

size\_t month;

size\_t day;

};

//преобразование даты в строку dd/mm/yyyy

std::string dateToString(Date);

//ФИО

struct FIO

{

char name[20];

char sername[20];

char fathername[20];

};

namespace worker

{

enum Gender{ MALE, FEMALE }; //пол

struct Worker

{

size\_t id; //id сотрудника

::FIO fio; //ФИО

Date birthday; //дата рождения

Gender gender; //пол

char profession[20];//профессия

size\_t experience; //опыт работы

size\_t rang; //ранг

size\_t factory; //id предприятия

size\_t sector; //номер сектора

double payment; // З/П

};

enum SortPriority

{

ID,

NAME,

SERNAME,

FATHERNAME,

BIRTHDAY,

GENDER,

PROFFESSION,

EXPERIENCE,

RANG,

FACTORY,

SECTOR,

PAYMENT

};

/\*сравнение сотрудников

\*@param

\* \_left - первый сотрудник

\* \_right - второй сотрудник

\* SortPriority - приоритет сортировки

\* @return

\* -1 - first, 0 equals, 1 - second

\*/

int compareWorker(const Worker& \_left, const Worker& \_right, SP sortPriority);

#define CMPFUNC(title) int(\*(title))(const Worker&, const Worker&, SP)

}

#endif

Worker.cpp

#include "Worker.h"

#include <string>

#include <cstring>

std::string dateToString(Date \_date)

{

std::string date(10, ' ');

date[0] = char(\_date.day / 10 + int('0'));

date[1] = char(\_date.day % 10 + int('0'));

date[2] = '/';

date[3] = char(\_date.month / 10 + int('0'));

date[4] = char(\_date.month % 10 + int('0'));

date[5] = '/';

date[6] = char(\_date.year / 1000 + int('0'));

\_date.year %= 1000;

date[7] = char(\_date.year / 100 + int('0'));

\_date.year %= 100;

date[8] = char(\_date.year / 10 + int('0'));

date[9] = char(\_date.year % 10 + int('0'));

return date;

}

namespace worker

{

int compareWorker(const Worker& \_left, const Worker& \_right,SP sortPriority)

{

switch (sortPriority)

{

case ID:

{

if (\_left.id == \_right.id)

return 0;

return \_left.id < \_right.id ? -1 : 1;

} break;

case NAME:

{

return -std::strcmp(\_left.fio.name, \_right.fio.name);

} break;

case SERNAME:

{

return -std::strcmp(\_left.fio.sername, \_right.fio.sername);

} break;

case FATHERNAME:

{

return -std::strcmp(\_left.fio.fathername, \_right.fio.fathername);

} break;

case BIRTHDAY:

{

LL firstDate = \_left.birthday.year \* 10000 + \_left.birthday.month \* 100 + \_left.birthday.day;

LL secondDate = \_right.birthday.year \* 10000 + \_right.birthday.month \* 100 + \_right.birthday.day;

if(firstDate == secondDate)

return 0;

else if(firstDate > secondDate)

return 1;

else

return -1;

} break;

case GENDER:

{

if (\_left.gender == \_right.gender)

return 0;

return \_left.gender == MALE ? -1 : 1;

} break;

case PROFFESSION:

{

return -std::strcmp(\_left.profession, \_right.profession);

} break;

case EXPERIENCE:

{

if (\_left.experience == \_right.experience)

return 0;

return \_left.experience < \_right.experience ? -1 : 1;

} break;

case RANG:

{

if (\_left.rang == \_right.rang)

return 0;

return \_left.rang < \_right.rang ? -1 : 1;

} break;

case FACTORY:

{

if (\_left.factory == \_right.factory)

return 0;

return \_left.factory < \_right.factory ? -1 : 1;

} break;

case SECTOR:

{

if (\_left.sector == \_right.sector)

return 0;

return \_left.sector < \_right.sector ? -1 : 1;

} break;

case PAYMENT:

{

if (\_left.payment == \_right.payment)

return 0;

return \_left.payment < \_right.payment ? -1 : 1;

} break;

}

}

}

Vector.h

#pragma once

#ifndef \_MYVECTOR\_H\_

#define \_MYVECTOR\_H\_

#include "Worker.h"

#include "macro.h"

namespace worker

{

/\*изменение размера вектора

\*@param

\* vector - указатель на вектор

\* \_size - текущий размер

\* \_newSize - новый размер

\* \_capacity - текущий занимаемый объем памяти

\*/

void resize(Worker\*\* vector, const size\_t& \_size,

const size\_t& \_newSize, size\_t& capacity);

/\*удаления части вектора

\*@param

\* vector - указатель на вектор

\* size - размер векора

\* \_first - первый удаляемый эл-т

\* \_last - последний удаляемый эл-т

\*/

void erase(Worker\*\* vector, size\_t& size,

const size\_t& \_first, const size\_t& \_last);

/\*удаления части вектора

\*@param

\* vector - указатель на вектор

\* size - размер векора

\* capacity - занимаемый объем памяти

\* \_new - новый работник

\* \_pos - номер позиции, на которую будет вставлен новый сотрудник

\*/

void insert(Worker\*\* vector, size\_t& size,

size\_t& capacity, const Worker& \_new, const size\_t& \_pos);

/\*сортировка вектора

\*@param

\* arr - указатель на вектор

\* \_left - левая граница сортировки

\* \_right - правая граница сортировки

\* \_priority - приоритет сортировки

\* cmpfunc - функция сравнения двух работников по приоритету

\*/

void quickSort(Worker\*\* arr, const int& \_left,

const int& \_right, SP \_priority, CMPFUNC(cmpfunc));

}

#endif

Vector.cpp

#include "Vector.h"

namespace worker

{

void resize(Worker\*\* vector, const size\_t& \_size, const size\_t& \_newSize, size\_t& capacity)

{

if (capacity >= \_newSize)

return;

Worker\* newVector = new Worker[\_newSize];

size\_t temp = \_size <= \_newSize ? \_size : \_newSize;

for (size\_t i = 0; i < temp; ++i)

newVector[i] = (\*vector)[i];

delete[](\*vector);

\*vector = newVector;

capacity = \_newSize;

}

void erase(Worker\*\* vector, size\_t& size, const size\_t& \_first, const size\_t& \_last)

{

size\_t delCount = \_last - \_first + 1;

for (size\_t i = \_last + 1; i < size; ++i)

(\*vector)[i - delCount] = (\*vector)[i];

size -= delCount;

}

void insert(Worker\*\* vector, size\_t& size, size\_t& capacity, const Worker& \_new, const size\_t& \_pos)

{

resize(&(\*vector), size, size + 2, capacity);

for (size\_t i = size; i > \_pos; --i)

(\*vector)[i] = (\*vector)[i - 1];

size++;

(\*vector)[\_pos] = \_new;

}

void quickSort(Worker\*\* arr, const int& \_left, const int& \_right, SP \_priority, CMPFUNC(cmpfunc))

{

int i = \_left, j = \_right;

Worker mid = (\*arr)[(\_left + \_right) / 2];

while (i <= j)

{

while (cmpfunc((\*arr)[i], mid, \_priority) < 0) i++;

while (cmpfunc((\*arr)[j], mid, \_priority) > 0) j--;

if (i <= j)

std::swap((\*arr)[i++],(\*arr)[j--]);

}

if (\_left < j)

quickSort(&(\*arr), \_left, j, \_priority, cmpfunc);

if (i < \_right)

quickSort(&(\*arr), i, \_right, \_priority, cmpfunc);

}

}

WorkerData.h

#pragma once

#ifndef \_WORKERSDATA\_H\_

#define \_WORKERSDATA\_H\_

#include "Vector.h"

#include "macro.h"

namespace worker

{

#define WD WorkersData

//данные о сотрудниках

struct WorkersData

{

SP sortPriority; //приоритет сортировки сотрудников

Worker\* workers; //вектор сотрудников

size\_t size; //кол-во сотрудников

size\_t capacity; //занимаемая вектором память/sizeof(worker)

};

/\*добавление нового сотрудника

\*@param

\* workers - стурктура хранения сотрудников

\* \_newWorker - новый сотрудник

\*/

void add(WD& workers, const Worker& \_newWorker);

/\*поиск сотрудника

\*@param

\* workers - стурктура хранения сотрудников

\* \_id - id сотрудника

\*@return

\* позиция в векторе или -1

\*/

long long search(const WD& workers, const size\_t& \_id);

/\*удаление сотрудника

\*@param

\* workers - стурктура хранения сотрудников

\* \_dataPos - позиция в векторе

\*/

void del(WD& workers, const size\_t& \_dataPos);

/\*сортировка сотрудников

\*@param

\* workers - стурктура хранения сотрудников

\* sortFlag - 0-убывание, 1-возрастание

\*/

void sort(WD& workers, int sortFlag = 0);

/\*освобождение памяти

\*@param

\* workers - стурктура хранения сотрудников

\*/

void clear(WD& workers);

}

#endif

WorkerData.cpp

#include "WorkersData.h"

namespace worker

{

void add(WorkersData& workers, const Worker& \_newWorker)

{

size\_t i;

for (i = 0; i < workers.size; ++i)

if (compareWorker(workers.workers[i], \_newWorker, workers.sortPriority) == 1)

break;

insert(&workers.workers, workers.size, workers.capacity, \_newWorker, i);

}

long long search(const WorkersData& \_workers, const size\_t& \_id)

{

for (size\_t i = 0; i < \_workers.size; ++i)

if (\_workers.workers[i].id == \_id)

return i;

return -1;

}

void del(WorkersData& workers, const size\_t& \_dataPos)

{

erase(&workers.workers, workers.size, \_dataPos, \_dataPos);

}

int cmpFuncInc(const Worker& \_left, const Worker& \_right, SP \_sp)

{

return -compareWorker(\_left, \_right, \_sp);

}

void sort(WorkersData& workers, int sortFlag)

{

if(workers.size > 1)

{

if(!sortFlag)

quickSort(&workers.workers, 0, workers.size - 1, workers.sortPriority, compareWorker);

else

quickSort(&workers.workers, 0, workers.size - 1, workers.sortPriority, cmpFuncInc);

}

}

void clear(WorkersData& \_workers)

{

delete[] \_workers.workers;

\_workers.workers = nullptr;

\_workers.capacity = 0;

\_workers.size = 0;

}

}

Factory.h

#pragma once

#ifndef \_FACTORY\_H\_

#define \_FACTORY\_H\_

#include "macro.h"

namespace factory

{

//количество сотрудников на предприятии

struct CountWorkers

{

size\_t all; //количесво всех сотрудников

size\_t male; //кол-во мужчин

size\_t female; //кол-во женщин

};

//процент работников на предприятии от общего

struct PersentCountWorkers

{

size\_t all;

size\_t male;

size\_t female;

};

struct Factory

{

size\_t number; //номер предприятия

CountWorkers count;

PersentCountWorkers persentCount;

};

//выбор поля сортировки

enum SortPriority

{

NUMBER,

ALL,

MALE,

FEMALE,

P\_ALL,

P\_MALE,

P\_FEMALE

};

//функция равнения двух полей по приоритету

int compareFactory(const Factory& \_left, const Factory& \_right, SP sortpriority);

#define CMPFUNC(title) int(\*(title))(const Factory&, const Factory&, SP)

}

#endif

Factory.cpp

#include"Factory.h"

namespace factory

{

int compareFactory(const Factory& \_left, const Factory& \_right, SP sp)

{

switch (sp)

{

case NUMBER:

{

if (\_left.number == \_right.number)

return 0;

return \_left.number > \_right.number ? -1 : 1;

} break;

case ALL:

{

if (\_left.count.all == \_right.count.all)

return 0;

return \_left.count.all > \_right.count.all ? -1 : 1;

} break;

case MALE:

{

if (\_left.count.male == \_right.count.male)

return 0;

return \_left.count.male > \_right.count.male ? -1 : 1;

} break;

case FEMALE:

{

if (\_left.count.female == \_right.count.female)

return 0;

return \_left.count.female > \_right.count.female ? -1 : 1;

} break;

case P\_ALL:

{

if (\_left.persentCount.all == \_right.persentCount.all)

return 0;

return \_left.persentCount.all > \_right.persentCount.all ? -1 : 1;

} break;

case P\_MALE:

{

if (\_left.persentCount.male == \_right.persentCount.male)

return 0;

return \_left.persentCount.male > \_right.persentCount.male ? -1 : 1;

} break;

case P\_FEMALE:

{

if (\_left.persentCount.female == \_right.persentCount.female)

return 0;

return \_left.persentCount.female > \_right.persentCount.female ? -1 : 1;

} break;

}

}

}

Tree.h

#pragma once

#ifndef \_WORKERSTATISTICSTREE\_H\_

#define \_WORKERSTATISTICSTREE\_H\_

#include "Factory.h"

#include "macro.h"

namespace factory

{

//дерево предприятий

struct FNode

{

Factory\* key;

FNode\* left;

FNode\* right;

};

/\*добавление предприятия

\*@param

\* root - указатель на корень дерева

\* \_key - добавляемое предприятие

\* \_priority - текущий приоритет сортировки дерева

\* cmpfunc - функция сравнение ключей

\*/

void add(FNode\*\* root, Factory\* \_key, SP \_priority, CMPFUNC(cmpfunc));

/\*поиск предприятия в дереве

\*@param

\* root - указатель на корень дерева

\* \_number - id предприятия

\*/

Factory\* search(FNode\* \_root, const size\_t& \_number);

/\*удаление предприятия из дерева

\*@param

\* root - указатель на корень дерева

\* \_number - id предприятия

\*@return

\* указатель на ключь или nullptr

\*/

void del(FNode\*\* root, const size\_t& \_number);

/\*перестраивание дерева по новому приоритету

\*@param

\* root - указатель на корень дерева

\* \_priority - приоритет

\* cmpfunc - функция сравнения ключей

\*/

void reBuild(FNode\*\* root, SP \_priority, CMPFUNC(cmpfunc));

/\*освобождение памяти

\*@param

\* root - указатель на корень дерева

\*/

void clear(FNode\*\* root);

}

#endif

Tree.cpp

#include "Tree.h"

namespace factory

{

void add(FNode\*\* root, Factory\* key, SP \_priority, CMPFUNC(cmpfunc))

{

if (\*root == nullptr)

{

\*root = new FNode;

(\*root)->key = key;

(\*root)->left = nullptr;

(\*root)->right = nullptr;

return;

}

int flag;

FNode\* temp = \*root;

while (true)

{

flag = cmpfunc( \*temp->key, \*key, \_priority);

if (!flag)

return;

if (flag == 1)

{

if (temp->right == nullptr)

{

temp->right = new FNode;

temp = temp->right;

temp->key = key;

temp->left = nullptr;

temp->right = nullptr;

return;

}

temp = temp->right;

}

else

{

if (temp->left == nullptr)

{

temp->left = new FNode;

temp = temp->left;

temp->key = key;

temp->left = nullptr;

temp->right = nullptr;

return;

}

temp = temp->left;

}

}

}

Factory\* search(FNode\* root, const size\_t& \_number)

{

while (true)

{

if (root == nullptr)

return nullptr;

if (root->key->number == \_number)

return root->key;

if (root->key->number < \_number)

root = root->right;

else

root = root->left;

}

}

void delNode(FNode\*\* node)

{

delete (\*node)->key;

delete (\*node);

\*node = nullptr;

}

void del(FNode\*\* root, const size\_t& \_number)

{

if (!(\*root)) return;

if ((\*root)->key->number > \_number)

{

del(&(\*root)->left, \_number);

return;

}

else if ((\*root)->key->number < \_number)

{

del(&(\*root)->right, \_number);

return;

}

FNode\* left = (\*root)->left;

FNode\* right = (\*root)->right;

delNode(root);

if (!left) \*root = right;

else if (!right) \*root = left;

else

{

\*root = left;

while (left->right)

{

left = left->right;

}

left->right = right;

}

}

void \_\_reBuild(FNode\* root, FNode\*\* newRoot, SP \_priority, CMPFUNC(cmpfunc))

{

if (root == nullptr)

return;

add(newRoot, root->key, \_priority, cmpfunc);

\_\_reBuild(root->left, newRoot, \_priority, cmpfunc);

\_\_reBuild(root->right, newRoot, \_priority, cmpfunc);

delete root;

}

void reBuild(FNode\*\* root, SP \_priority, CMPFUNC(cmpfunc))

{

if (\*root == nullptr)

return;

FNode\* newRoot = nullptr;

\_\_reBuild(\*root, &newRoot, \_priority, cmpfunc);

\*root = newRoot;

}

void clear(FNode\*\* root)

{

if (\*root == nullptr)

return;

clear(&((\*root)->left));

clear(&((\*root)->right));

delNode(root);

}

}

FactoryData.h

#pragma once

#ifndef \_FACTORYESDATA\_H\_

#define \_FACTORYESDATA\_H\_

#include "Tree.h"

#include "macro.h"

namespace factory

{

#define FD FactoryData

//данные о предприятиях

struct FactoryData

{

SP sortPriority; //приоритет сортировки

FNode\* root; //указатель на корень дерева

size\_t size; //кол-во эл-тов дерева

};

/\*добавление предприятия в FactoryData

\*@param

\* factoryes - структура хранения предприятий

\* \_factory - добавляемое предприятие

\*/

void add(FD& factoryes, Factory\* \_factory);

/\*поиск предприятия по id

\*@param

\* factoryes - структура хранения предприятий

\* \_id - id предприятия

\*@return

\* указатель на преприятие or nullptr

\*/

Factory\* search(const FD& \_factoryes, const size\_t& \_id);

/\*поиск предприятия по id

\*@param

\* factoryes - структура хранения предприятий

\* \_id - id предприятия

\*/

void del(FD& factoryes, const size\_t& \_id);

/\*сортировка предприятий

\*@param

\* factoryes - структура хранения предприятий

\* cmpFlag - 0(по убыванию), 1 (по возрастанию)

\*/

void sort(FD& factoryes, int cmpFlag = 0);

/\*освобождение памяти

\*@param

\* factoryes - структура хранения предприятий

\*/

void clearData(FD& factoryes);

}

#endif

FactoryData.cpp

#include "FactoryesData.h"

namespace factory

{

void add(FD& factoryes, Factory\* \_factory)

{

add(&factoryes.root, \_factory,factoryes.sortPriority,compareFactory);

factoryes.size++;

}

Factory\* search(const FD& \_factoryes, const size\_t& \_id)

{

return search(\_factoryes.root, \_id);

}

void del(FD& factoryes, const size\_t& \_id)

{

del(&factoryes.root, \_id);

factoryes.size--;

}

int cmpFuncInc(const Factory& \_left, const Factory& \_right, SP \_sp)

{

return -compareFactory(\_left, \_right, \_sp);

}

void sort(FD& factoryes, int cmpFlag)

{

if (!cmpFlag)

//dec sort

reBuild(&factoryes.root, factoryes.sortPriority, compareFactory);

else

//inc sort

reBuild(&factoryes.root, factoryes.sortPriority, cmpFuncInc);

}

void clearData(FD& factoryes)

{

factory::clear(&factoryes.root);

factoryes.size = 0;

}

}

Data.h

#pragma once

#ifndef \_DATA\_H\_

#define \_DATA\_H\_

#include "WorkersData.h"

#include "FactoryesData.h"

#include "macro.h"

#include <string>

namespace data

{

struct Data

{

worker::WD workersData;

factory::FD factoryesData;

size\_t all;

size\_t male;

size\_t female;

};

/\*добавление нового работника

\*@param

\* \_data - дынные о всех сотрудниках

\* \_newWorker - добавляемый работник

\*/

void addWorker(Data& \_data, const worker::Worker& \_newWorker);

/\*удаление сотрудника из Data

\* по его позиции в списке

\*@param

\* \_data - дынные о всех сотрудниках

\* \_dataPos - розиция сотрудника в списке

\*/

void delWorker(Data& \_data, const size\_t& \_dataPos);

/\*сохранение данных в двоичный файл

\* @param

\* \_data - данные о всех сотрудниках

\* \_path - путь к файлу сохранения

\*/

void saveBinDataFile(const Data& \_data, const std::string& \_path);

/\*загрузка данных из двоичного файла

\* @param

\* \_data - данные о всех сотрудниках

\* \_path - путь к файлу загрузки

\*/

void loadBinDataFile(Data& \_data, const std::string& \_path);

/\*вывод таблицы предприятий в текстовый файл

\* @param

\* \_data - данные о всех сотрудниках

\* \_path - путь к файлу сохранения

\*/

void printFactoryTableTextFile(const Data& \_data, const std::string& \_path);

/\*освобождение памяти

\* @param

\* \_data - данные о всех сотрудниках

\*/

void clearData(Data& \_data);

/\*инициализация данных

\* @param

\* \_data - данные о всех сотрудниках

\*/

void initialize(Data& \_data);

}

#endif

Data.cpp

#include <fstream>

#include <iostream>

#include <iomanip>

#include "Data.h"

namespace data

{

void \_\_prepare(Data& \_data)

{

if (\_data.factoryesData.sortPriority != factory::SP::NUMBER)

{

\_data.factoryesData.sortPriority = factory::SP::NUMBER;

factory::sort(\_data.factoryesData);

}

}

void \_\_persentCalculate(factory::FNode\* root, const int& male, const int& female, const int& all)

{

if (root == nullptr)

return;

root->key->persentCount.all = bool(root->key->count.all) ? int(100.0 \* double(root->key->count.all) / double(all)) : 0;

root->key->persentCount.male = bool(root->key->count.male) ? int(100.0 \* double(root->key->count.male) / double(male)) : 0;

root->key->persentCount.female = bool(root->key->count.female) ? int(100.0 \* double(root->key->count.female) / double(female)) : 0;

\_\_persentCalculate(root->left, male, female, all);

\_\_persentCalculate(root->right, male, female, all);

}

void addWorker(Data& \_data, const worker::Worker& \_newWorker)

{

worker::add(\_data.workersData, \_newWorker);

\_\_prepare(\_data);

factory::Factory\* factory = factory::search(\_data.factoryesData, \_newWorker.factory);

if (factory == nullptr)

{

factory = new factory::Factory;

factory->number = \_newWorker.factory;

factory->count.all = 0;

factory->count.male = 0;

factory->count.female = 0;

factory->persentCount.all = 0;

factory->persentCount.male = 0;

factory->persentCount.female = 0;

factory::add(\_data.factoryesData, factory);

}

\_data.all++;

if (\_newWorker.gender == worker::MALE)

{

\_data.male++;

factory->count.all++;

factory->count.male++;

}

else

{

\_data.female++;

factory->count.all++;

factory->count.female++;

}

\_\_persentCalculate(\_data.factoryesData.root, \_data.male, \_data.female, \_data.all);

}

void delWorker(Data& \_data, const size\_t& \_dataPos)

{

worker::Worker& worker = \_data.workersData.workers[\_dataPos];

\_\_prepare(\_data);

factory::Factory\* factory = factory::search(\_data.factoryesData, worker.factory);

worker::Gender gender = worker.gender;

if (factory->count.all > 1)

{

factory->count.all--;

if (gender == worker::MALE)

factory->count.male--;

else

factory->count.female--;

}

else

{

factory::del(\_data.factoryesData, factory->number);

}

\_data.all--;

if(gender == worker::MALE)

\_data.male--;

else

\_data.female--;

worker::del(\_data.workersData, \_dataPos);

\_\_persentCalculate(\_data.factoryesData.root, \_data.male, \_data.female, \_data.all);

}

void saveBinDataFile(const Data& \_data, const std::string& \_path)

{

std::ofstream fout(\_path, std::ios::binary);

int count = \_data.workersData.size;

fout.write(reinterpret\_cast<char\*>(&count), sizeof(int));

for(int i = 0; i < count; ++i)

fout.write(reinterpret\_cast<char\*>(&\_data.workersData.workers[i]), sizeof(worker::Worker));

fout.close();

}

void loadBinDataFile(Data& \_data, const std::string& \_path)

{

std::ifstream fin(\_path, std::ios::binary);

data::clearData(\_data);

\_data.workersData.sortPriority = worker::SP::ID;

\_data.factoryesData.sortPriority = factory::SP::NUMBER;

int countWorkers;

fin.read(reinterpret\_cast<char\*>(&countWorkers), sizeof(int));

worker::Worker worker;

for(int i = 0; i < countWorkers; ++i)

{

fin.read(reinterpret\_cast<char\*>(&worker), sizeof(worker::Worker));

data::addWorker(\_data, worker);

}

fin.close();

}

void \_\_print(const factory::FNode\* root, std::ofstream& fout)

{

if (root == nullptr)

return;

\_\_print(root->left, fout);

fout << std::setw(6) << root->key->number << '|';

fout << std::setw(8) << root->key->count.male << '|';

fout << std::setw(8) << root->key->count.female << '|';

fout << std::setw(8) << root->key->count.all << '|';

fout << std::setw(8) << root->key->persentCount.male << '|';

fout << std::setw(8) << root->key->persentCount.female << '|';

fout << std::setw(8) << root->key->persentCount.all;

fout << std::endl;

\_\_print(root->right, fout);

}

void printFactoryTableTextFile(const Data& \_data, const std::string& \_path)

{

std::ofstream fout(\_path);

fout.setf(std::ios::left);

fout.setf(std::ios::fixed);

fout.precision(0);

fout << " | Count | Percent" << std::endl;

fout << " Id | Male | Female | All | Male | Female | All" << std::endl;

\_\_print(\_data.factoryesData.root, fout);

fout.close();

}

void clearData(Data& \_data)

{

factory::clearData(\_data.factoryesData);

worker::clear(\_data.workersData);

\_data.all = 0;

\_data.male = 0;

\_data.female = 0;

}

void initialize(Data& \_data)

{

\_data.all = 0;

\_data.female = 0;

\_data.male = 0;

\_data.workersData.workers = nullptr;

\_data.workersData.size = 0;

\_data.workersData.capacity = 0;

\_data.workersData.sortPriority = worker::SortPriority::ID;

\_data.factoryesData.root = nullptr;

\_data.factoryesData.size = 0;

\_data.factoryesData.sortPriority = factory::SortPriority::NUMBER;

}

}

MainWindow.h

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include "Data.h"

#include "searchdialog.h"

#include <QMainWindow>

#include <QAction>

#include <QMenu>

#include <QTableWidget>

#include <QRadioButton>

#include <QPushButton>

#include <QLineEdit>

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

MainWindow(QWidget \*parent = 0);

~MainWindow();

private:

void keyPressEvent(QKeyEvent \*);

private slots:

void tableChanged();

void addClicked();

void editClicked();

void addWorker(const worker::Worker&, int);

void changeSelectedWorker(const worker::Worker&, int);

void removeSelectedWorkers();

void saveClicked();

void saveFTableClicked();

void openClicked();

void setWorkersTableSortPriority(int);

void setFactoryesTableSortPriority(int);

void itemSelectionChanged();

void enableTrue();

void searchClicked();

void showSearchResult(int);

private:

void printWorkersTable();

void printTreeTable(factory::FNode\*, int&);

void printFactoryesTable();

//Data

data::Data workers;

int sortDirection;

//menu bar

QMenu\* mainMenu;

QAction\* exit;

QAction\* open;

QAction\* save;

QAction\* saveFactoryesTable;

QWidget\* centralWidget;

SearchDialog\* searchDialog;

//tables

QTableWidget\* workersTable;

QTableWidget\* factoryesTable;

QRadioButton\* workersView;

QRadioButton\* factoryesView;

//edit buttons

QPushButton\* add;

QPushButton\* edit;

QPushButton\* search;

QPushButton\* del;

};

#endif // MAINWINDOW\_H

MainWindow.cpp

#include "mainwindow.h"

#include "workerdialog.h"

#include <QMessageBox>

#include <QApplication>

#include <QMenuBar>

#include <QFileDialog>

#include <QList>

#include <QQueue>

#include <QHBoxLayout>

#include <QVBoxLayout>

#include <QTableWidgetItem>

#include <QHeaderView>

#include <QKeyEvent>

MainWindow::MainWindow(QWidget \*parent)

: QMainWindow(parent)

{

this->resize(maximumWidth(), maximumHeight());

mainMenu = new QMenu(QString("&Menu"), this);

QList < QAction\* > menuList;

open = new QAction(QString("Open"), this);

save = new QAction(QString("Save"),this);

saveFactoryesTable = new QAction(QString("Save factoryes table .txt"),this);

exit = new QAction(QString("Exit"),this);

menuList.append(open);

menuList.append(save);

menuList.append(saveFactoryesTable);

menuList.append(exit);

mainMenu->addActions(menuList);

menuBar()->addMenu(mainMenu);

QHBoxLayout\* topLay = new QHBoxLayout(this);

workersView = new QRadioButton("Workers", this);

workersView->setChecked(true);

factoryesView = new QRadioButton("Factoryes", this);

topLay->addWidget(workersView);

topLay->addWidget(factoryesView);

topLay->addStretch();

QVBoxLayout\* leftLay = new QVBoxLayout(this);

add = new QPushButton("&Add", this);

edit = new QPushButton("&Edit", this);

search = new QPushButton("Search", this);

del = new QPushButton("Delete", this);

edit->setEnabled(false);

del->setEnabled(false);

leftLay->addWidget(add);

leftLay->addWidget(edit);

leftLay->addWidget(search);

leftLay->addWidget(del);

leftLay->addStretch();

QHBoxLayout\* botLay = new QHBoxLayout(this);

workersTable = new QTableWidget(0, 12, this);

workersTable->horizontalHeader()->setStretchLastSection(true);

workersTable->setSelectionBehavior(QAbstractItemView::SelectRows);

QStringList workersTableHeader;

workersTableHeader << "Id" << "Name" << "Sername"<<

"Fathername" << "Birthday" <<

"Gender" << "Profession" <<

"Experience" << "Rang" <<

"Factory" << "Sector" << "Payment";

workersTable->setHorizontalHeaderLabels(workersTableHeader);

factoryesTable = new QTableWidget(0, 7, this);

factoryesTable->horizontalHeader()->setStretchLastSection(true);

factoryesTable->setSelectionBehavior(QAbstractItemView::SelectRows);

QStringList factoryesTableHeader;

factoryesTableHeader << "ID" << "Male" << "Female" << "All" << "Male(%)" << "Female(%)" << "All(%)";

factoryesTable->setHorizontalHeaderLabels(factoryesTableHeader);

botLay->addLayout(leftLay);

botLay->addWidget(workersTable);

botLay->addWidget(factoryesTable);

factoryesTable->setVisible(false);

QVBoxLayout\* mainLay = new QVBoxLayout(this);

mainLay->addLayout(topLay);

mainLay->addLayout(botLay);

centralWidget = new QWidget(this);

centralWidget->setLayout(mainLay);

this->setCentralWidget(centralWidget);

//Data init

data::initialize(workers);

sortDirection = 0;

QObject::connect(exit, SIGNAL(triggered(bool)),

this, SLOT(close()));

QObject::connect(workersView,SIGNAL(clicked(bool)),

this, SLOT(tableChanged()));

QObject::connect(factoryesView,SIGNAL(clicked(bool)),

this, SLOT(tableChanged()));

QObject::connect(add, SIGNAL(clicked(bool)),

this, SLOT(addClicked()));

QObject::connect(edit, SIGNAL(clicked(bool)),

this, SLOT(editClicked()));

QObject::connect(del, SIGNAL(clicked(bool)),

this, SLOT(removeSelectedWorkers()));

QObject::connect(save, SIGNAL(triggered(bool)),

this, SLOT(saveClicked()));

QObject::connect(open, SIGNAL(triggered(bool)),

this, SLOT(openClicked()));

QObject::connect(workersTable->horizontalHeader(),SIGNAL(sectionClicked(int)),

this, SLOT(setWorkersTableSortPriority(int)));

QObject::connect(factoryesTable->horizontalHeader(),SIGNAL(sectionClicked(int)),

this, SLOT(setFactoryesTableSortPriority(int)));

QObject::connect(workersTable, SIGNAL(itemSelectionChanged()),

this, SLOT(itemSelectionChanged()));

QObject::connect(search, SIGNAL(clicked(bool)),

this, SLOT(searchClicked()));

QObject::connect(saveFactoryesTable, SIGNAL(triggered(bool)),

this, SLOT(saveFTableClicked()));

}

MainWindow::~MainWindow()

{

delete mainMenu;

delete exit;

delete save;

delete saveFactoryesTable;

delete workersTable;

delete factoryesTable;

delete workersView;

delete factoryesView;

delete add;

delete edit;

delete del;

data::clearData(workers);

}

void MainWindow::keyPressEvent(QKeyEvent\* event)

{

switch(event->key())

{

case Qt::Key\_Delete:

{

if(del->isEnabled())

this->removeSelectedWorkers();

} break;

case Qt::Key\_F:

{

if (QApplication::keyboardModifiers() && Qt::ControlModifier)

this->searchClicked();

} break;

case Qt::Key\_S:

{

if (QApplication::keyboardModifiers() && Qt::ControlModifier)

this->saveClicked();

} break;

}

}

void MainWindow::tableChanged()

{

if(workersView->isChecked())

{

workersTable->setVisible(true);

add->setVisible(true);

edit->setVisible(true);

del->setVisible(true);

search->setVisible(true);

factoryesTable->setVisible(false);

}

else

{

workersTable->setVisible(false);

workersTable->clearSelection();

add->setVisible(false);

search->setVisible(false);

edit->setVisible(false);

del->setVisible(false);

factoryesTable->setVisible(true);

}

}

void MainWindow::addClicked()

{

WorkerDialog\* addDialog = new WorkerDialog(this);

this->setEnabled(false);

addDialog->setEnabled(true);

QObject::connect(addDialog,SIGNAL(okClicked(worker::Worker, int)),

this, SLOT(addWorker(worker::Worker, int)));

QObject::connect(addDialog, SIGNAL(rejected()),

this, SLOT(enableTrue()));

addDialog->show();

addDialog->exec();

}

void MainWindow::editClicked()

{

this->setEnabled(false);

int workerPos = this->workersTable->selectedRanges().at(0).bottomRow();

WorkerDialog\* editDialog = new WorkerDialog(workers.workersData.workers[workerPos], this);

editDialog->setEnabled(true);

QObject::connect(editDialog, SIGNAL(okClicked(worker::Worker, int)),

this, SLOT(changeSelectedWorker(worker::Worker, int)));

QObject::connect(editDialog, SIGNAL(okClicked(worker::Worker, int)),

editDialog, SLOT(close()));

QObject::connect(editDialog, SIGNAL(rejected()),

this, SLOT(enableTrue()));

editDialog->show();

editDialog->exec();

}

void MainWindow::addWorker(const worker::Worker& newWorker, int)

{

if (worker::search(workers.workersData, newWorker.id) == -1)

{

data::addWorker(workers, newWorker);

printWorkersTable();

printFactoryesTable();

}

else

{

QMessageBox\* result = new QMessageBox(this);

result->setWindowTitle("Adding error");

result->setText("This ID are reserved");

result->show();

result->exec();

}

}

void MainWindow::changeSelectedWorker(const worker::Worker& newWorker, int prevId)

{

if(worker::search(workers.workersData, newWorker.id) == -1 || prevId == newWorker.id)

{

data::delWorker(workers, this->workersTable->selectedRanges().at(0).bottomRow());

data::addWorker(workers, newWorker);

printWorkersTable();

printFactoryesTable();

}

else

{

QMessageBox\* result = new QMessageBox(this);

result->setWindowTitle("Change error");

result->setText("This ID are reserved");

result->show();

result->exec();

}

}

void MainWindow::printWorkersTable()

{

int rowCount = workers.workersData.size;

workersTable->setRowCount(rowCount);

worker::Worker nextWorker;

QTableWidgetItem temp[12];

for(int i = 0; i < 12; ++i)

temp[i].setFlags(temp[i].flags() ^ Qt::ItemIsEditable);

for(int i = 0; i < rowCount; ++i)

{

nextWorker = workers.workersData.workers[i];

temp[0].setText(tr("%1").arg(nextWorker.id));

temp[1].setText(QString(nextWorker.fio.name));

temp[2].setText(QString(nextWorker.fio.sername));

temp[3].setText(QString(nextWorker.fio.fathername));

temp[4].setText(QString(dateToString(nextWorker.birthday).c\_str()));

temp[5].setText(QString(((nextWorker.gender == worker::MALE) ? ("Male") : ("Female"))));

temp[6].setText(QString(nextWorker.profession));

temp[7].setText(tr("%1").arg(nextWorker.experience));

temp[8].setText(tr("%1").arg(nextWorker.rang));

temp[9].setText(tr("%1").arg(nextWorker.factory));

temp[10].setText(tr("%1").arg(nextWorker.sector));

temp[11].setText(tr("%1").arg(nextWorker.payment, 0, 'f', 2));

for(int j = 0; j < 12; ++j)

workersTable->setItem(i, j, new QTableWidgetItem(temp[j]));

}

}

void MainWindow::printTreeTable(factory::FNode\* root, int& pos)

{

if(root == nullptr)

return;

printTreeTable(root->left, pos);

QTableWidgetItem temp[7];

for(int i = 0; i < 7; ++i)

temp[i].setFlags(temp[i].flags() ^ Qt::ItemIsEditable);

temp[0].setText(tr("%1").arg(root->key->number));

temp[1].setText(tr("%1").arg(root->key->count.male));

temp[2].setText(tr("%1").arg(root->key->count.female));

temp[3].setText(tr("%1").arg(root->key->count.all));

temp[4].setText(tr("%1").arg(root->key->persentCount.male));

temp[5].setText(tr("%1").arg(root->key->persentCount.female));

temp[6].setText(tr("%1").arg(root->key->persentCount.all));

for(int i = 0; i < 7; ++i)

factoryesTable->setItem(pos, i, new QTableWidgetItem(temp[i]));

pos++;

printTreeTable(root->right, pos);

}

void MainWindow::printFactoryesTable()

{

this->factoryesTable->setRowCount(workers.factoryesData.size);

int pos = 0;

if(workers.factoryesData.size)

this->printTreeTable(workers.factoryesData.root, pos);

}

void MainWindow::removeSelectedWorkers()

{

auto selectedWorkers = this->workersTable->selectedRanges();

for(int i = selectedWorkers.size() - 1; i > -1; --i)

{

for(int j = selectedWorkers.at(i).bottomRow(); j >= selectedWorkers.at(i).topRow(); --j)

data::delWorker(workers, j);

}

printWorkersTable();

printFactoryesTable();

}

void MainWindow::saveClicked()

{

if (workers.all == 0)

{

QMessageBox\* result = new QMessageBox(this);

result->setWindowTitle("Save error");

result->setText("Data is empty");

result->show();

result->exec();

return;

}

this->setEnabled(false);

QString path = QFileDialog::getSaveFileName(this,QString("Select path"),

QDir::homePath(), QString("Data file (\*.dat)"));

this->setEnabled(true);

if(!path.length())

return;

data::saveBinDataFile(workers, path.toStdString());

}

void MainWindow::saveFTableClicked()

{

if (workers.all == 0)

{

QMessageBox\* result = new QMessageBox(this);

result->setWindowTitle("Save error");

result->setText("Data is empty");

result->show();

result->exec();

return;

}

this->setEnabled(false);

QString path = QFileDialog::getSaveFileName(this,QString("Select path"),

QDir::homePath(), QString("Text file (\*.txt)"));

this->setEnabled(true);

if(!path.length())

return;

data::printFactoryTableTextFile(workers, path.toStdString());

}

void MainWindow::openClicked()

{

QString path = QFileDialog::getOpenFileName(0, QString("Select path"),

QDir::homePath(), QString("Data files (\*.dat);;All files(\*.\*)"));

if(!path.length())

return;

data::clearData(workers);

data::initialize(workers);

data::loadBinDataFile(workers, path.toStdString());

this->printFactoryesTable();

this->printWorkersTable();

}

void MainWindow::setFactoryesTableSortPriority(int priority)

{

factory::SortPriority prevPriority = workers.factoryesData.sortPriority;

switch(priority)

{

case 0: workers.factoryesData.sortPriority = factory::NUMBER; break;

case 1: workers.factoryesData.sortPriority = factory::MALE; break;

case 2: workers.factoryesData.sortPriority = factory::FEMALE; break;

case 3: workers.factoryesData.sortPriority = factory::ALL; break;

case 4: workers.factoryesData.sortPriority = factory::P\_MALE; break;

case 5: workers.factoryesData.sortPriority = factory::P\_FEMALE; break;

case 6: workers.factoryesData.sortPriority = factory::P\_ALL; break;

}

if(workers.factoryesData.sortPriority != prevPriority || sortDirection == 1)

{

sortDirection = 0;

factory::sort(workers.factoryesData, 0);

}

else

{

sortDirection = 1;

factory::sort(workers.factoryesData, 1);

}

this->printFactoryesTable();

}

void MainWindow::setWorkersTableSortPriority(int priority)

{

worker::SortPriority prevPriority = workers.workersData.sortPriority;

switch(priority)

{

case 0: workers.workersData.sortPriority = worker::ID; break;

case 1: workers.workersData.sortPriority = worker::NAME; break;

case 2: workers.workersData.sortPriority = worker::SERNAME; break;

case 3: workers.workersData.sortPriority = worker::FATHERNAME; break;

case 4: workers.workersData.sortPriority = worker::BIRTHDAY; break;

case 5: workers.workersData.sortPriority = worker::GENDER; break;

case 6: workers.workersData.sortPriority = worker::PROFFESSION; break;

case 7: workers.workersData.sortPriority = worker::EXPERIENCE; break;

case 8: workers.workersData.sortPriority = worker::RANG; break;

case 9: workers.workersData.sortPriority = worker::FACTORY; break;

case 10: workers.workersData.sortPriority = worker::SECTOR; break;

case 11: workers.workersData.sortPriority = worker::PAYMENT; break;

}

if(workers.workersData.sortPriority != prevPriority || sortDirection == 1)

{

sortDirection = 0;

worker::sort(workers.workersData);

}

else

{

sortDirection = 1;

worker::sort(workers.workersData, 1);

}

this->printWorkersTable();

}

void MainWindow::itemSelectionChanged()

{

int selectedItems = workersTable->selectedItems().count();

if(!selectedItems)

{

del->setEnabled(false);

edit->setEnabled(false);

return;

}

del->setEnabled(true);

if(selectedItems > 15)

edit->setEnabled(false);

else

edit->setEnabled(true);

}

void MainWindow::enableTrue()

{

this->setEnabled(true);

}

void MainWindow::searchClicked()

{

this->setEnabled(false);

searchDialog = new SearchDialog(&workers.workersData, this);

QObject::connect(searchDialog, SIGNAL(rejected()),

this, SLOT(enableTrue()));

QObject::connect(searchDialog, SIGNAL(searchClicked(int)),

this, SLOT(showSearchResult(int)));

QObject::connect(searchDialog, SIGNAL(searchClicked(int)),

searchDialog, SLOT(close()));

searchDialog->show();

searchDialog->exec();

}

void MainWindow::showSearchResult(int pos)

{

searchDialog->close();

delete searchDialog;

if (pos == -1)

{

QMessageBox\* result = new QMessageBox(this);

result->setWindowTitle("Search result");

result->setText("Null result");

result->show();

result->exec();

this->setEnabled(true);

return;

}

this->workersTable->clearSelection();

this->workersTable->selectRow(pos);

this->editClicked();

}

SearchDialog.h

#ifndef SEARCHDIALOG\_H

#define SEARCHDIALOG\_H

#include <QDialog>

#include "WorkersData.h"

namespace Ui {

class SearchDialog;

}

class SearchDialog : public QDialog

{

Q\_OBJECT

public:

SearchDialog(const worker::WorkersData\* data, QWidget \*parent = 0);

~SearchDialog();

private slots:

void search();

void argChanged();

signals:

void searchClicked(int);

private:

const worker::WorkersData\* data\_;

Ui::SearchDialog \*ui;

};

#endif // SEARCHDIALOG\_H

SearchDialog.h

#include "searchdialog.h"

#include "ui\_searchdialog.h"

SearchDialog::SearchDialog(const worker::WorkersData\* data, QWidget \*parent) :

QDialog(parent),

ui(new Ui::SearchDialog)

{

ui->setupUi(this);

data\_ = data;

QObject::connect(ui->id, SIGNAL(textChanged(QString)),

this, SLOT(argChanged()));

QObject::connect(ui->search, SIGNAL(clicked(bool)),

this, SLOT(search()));

QObject::connect(ui->cancle, SIGNAL(clicked(bool)),

this, SLOT(close()));

}

SearchDialog::~SearchDialog()

{

delete ui;

}

void SearchDialog::search()

{

int searchID = ui->id->text().toInt();

emit searchClicked(worker::search(\*data\_, searchID));

}

void SearchDialog::argChanged()

{

if(ui->id->text().length())

ui->search->setEnabled(true);

else

ui->search->setEnabled(false);

}

workerdialog.h

#ifndef WORKERDIALOG\_H

#define WORKERDIALOG\_H

#include "Worker.h"

#include <QDialog>

namespace Ui {

class WorkerDialog;

}

class WorkerDialog : public QDialog

{

Q\_OBJECT

public:

WorkerDialog(QWidget \*parent = 0);

WorkerDialog(const worker::Worker& account, QWidget\* parent = 0);

~WorkerDialog();

private slots:

void argChanged();

void ok();

signals:

void okClicked(worker::Worker, int);

private:

int prevId;

Ui::WorkerDialog \*ui;

};

#endif // WORKERDIALOG\_H

workerdialog.cpp

#include "workerdialog.h"

#include "ui\_workerdialog.h"

WorkerDialog::WorkerDialog(QWidget \*parent) :

QDialog(parent),

ui(new Ui::WorkerDialog)

{

ui->setupUi(this);

ui->name->clear();

ui->sername->clear();

ui->fathername->clear();

ui->factoryId->clear();

ui->id->clear();

ui->payment->clear();

ui->profession->clear();

ui->payment->setValidator(new QRegExpValidator(QRegExp("^([1-9][0-9]\*|0)(\\.)[0-9]\*|0"),this));

prevId = -1;

QObject::connect(ui->cancle, SIGNAL(clicked(bool)),

this, SLOT(close()));

QObject::connect(ui->ok, SIGNAL(clicked(bool)),

this, SLOT(ok()));

QObject::connect(ui->name, SIGNAL(textChanged(QString)),

this, SLOT(argChanged()));

QObject::connect(ui->sername, SIGNAL(textChanged(QString)),

this, SLOT(argChanged()));

QObject::connect(ui->fathername, SIGNAL(textChanged(QString)),

this, SLOT(argChanged()));

QObject::connect(ui->id, SIGNAL(textChanged(QString)),

this, SLOT(argChanged()));

QObject::connect(ui->birthday, SIGNAL(textChanged(QString)),

this, SLOT(argChanged()));

QObject::connect(ui->profession, SIGNAL(textChanged(QString)),

this, SLOT(argChanged()));

QObject::connect(ui->factoryId, SIGNAL(textChanged(QString)),

this, SLOT(argChanged()));

QObject::connect(ui->sectorId, SIGNAL(textChanged(QString)),

this, SLOT(argChanged()));

QObject::connect(ui->payment, SIGNAL(textChanged(QString)),

this, SLOT(argChanged()));

QObject::connect(ui->male, SIGNAL(clicked(bool)),

this, SLOT(argChanged()));

QObject::connect(ui->female, SIGNAL(clicked(bool)),

this, SLOT(argChanged()));

QObject::connect(ui->rang, SIGNAL(valueChanged(int)),

this, SLOT(argChanged()));

QObject::connect(ui->experience, SIGNAL(valueChanged(int)),

this, SLOT(argChanged()));

}

WorkerDialog::WorkerDialog(const worker::Worker& account, QWidget\* parent)

: WorkerDialog(parent)

{

prevId = account.id;

ui->ok->setText("Ok");

ui->name->setText(QString(account.fio.name));

ui->sername->setText(QString(account.fio.sername));

ui->fathername->setText(QString(account.fio.fathername));

ui->profession->setText(QString(account.profession));

ui->birthday->setText(QString(dateToString(account.birthday).c\_str()));

ui->id->setText(tr("%1").arg(account.id));

ui->experience->setValue(account.experience);

ui->factoryId->setText(tr("%1").arg(account.factory));

ui->sectorId->setText(tr("%1").arg(account.sector));

if(account.gender == worker::MALE)

{

ui->male->setChecked(true);

ui->female->setChecked(false);

}

else

{

ui->male->setChecked(false);

ui->female->setChecked(true);

}

ui->payment->setText(tr("%1").arg(account.payment, 0 , 'f', 2));

ui->rang->setValue(account.rang);

ui->ok->setEnabled(false);

}

WorkerDialog::~WorkerDialog()

{

delete ui;

}

void WorkerDialog::argChanged()

{

if(ui->name->text().isEmpty() ||

ui->sername->text().isEmpty() ||

ui->fathername->text().isEmpty() ||

ui->id->text().isEmpty() ||

ui->factoryId->text().isEmpty() ||

ui->payment->text().isEmpty() ||

ui->profession->text().isEmpty() ||

(ui->birthday->text().length() != 10))

{

ui->ok->setEnabled(false);

}

else

{

ui->ok->setEnabled(true);

}

}

void WorkerDialog::ok()

{

ui->ok->setEnabled(false);

worker::Worker newWorker;

memset(newWorker.fio.name, 0, sizeof(newWorker.fio.name));

memset(newWorker.fio.sername, 0, sizeof(newWorker.fio.sername));

memset(newWorker.fio.fathername, 0, sizeof(newWorker.fio.name));

memset(newWorker.profession, 0, sizeof(newWorker.profession));

std::string buffer;

buffer = ui->name->text().toStdString();

for(int i = 0; i < ui->name->text().length(); ++i)

newWorker.fio.name[i] = buffer[i];

buffer = ui->sername->text().toStdString();

for(int i = 0; i < ui->sername->text().length(); ++i)

newWorker.fio.sername[i] = buffer[i];

buffer = ui->fathername->text().toStdString();

for(int i = 0; i < ui->fathername->text().length(); ++i)

newWorker.fio.fathername[i] = buffer[i];

buffer = ui->profession->text().toStdString();

for(int i = 0; i < ui->profession->text().length(); ++i)

newWorker.profession[i] = buffer[i];

newWorker.birthday.day = QString(ui->birthday->text().toStdString().substr(0,2).c\_str()).toInt();

newWorker.birthday.month = QString(ui->birthday->text().toStdString().substr(3,2).c\_str()).toInt();

newWorker.birthday.year = QString(ui->birthday->text().toStdString().substr(6,4).c\_str()).toInt();

newWorker.experience = ui->experience->value();

newWorker.rang = ui->rang->value();

newWorker.factory = ui->factoryId->text().toInt();

newWorker.sector = ui->sectorId->text().toInt();

newWorker.gender = ui->male->isChecked() ? worker::MALE : worker::FEMALE;

newWorker.id = ui->id->text().toInt();

newWorker.payment = ui->payment->text().toDouble();

emit okClicked(newWorker, prevId);

}