

Государственное образовательное учреждение
высшего профессионального образования
«Санкт-Петербургский университет телекоммуникаций
им. проф. М.А. Бонч-Бруевича»

отм.


Курсовая работа по дисциплине:
«Объектно-ориентированное программирование»
Вариант 12

Выполнила:
Студентка 2 курса, группа
ИКПИ-51
Павшева Мария Владимировна

Преподаватель:
Козин Станислав Васильевич

Санкт-Петербург
2016

Оглавление

1. ПОСТАНОВКА ЗАДАЧИ	4
2. ВЫБОР И ОБОСНОВАНИЕ ТИПА РАЗРАБАТЫВАЕМОГО КОНТЕЙНЕРА	5
3. РАЗРАБОТКА СТРУКТУРЫ ПРОГРАММЫ.....	6
4. ВЫБОР И ОБОСНОВАНИЕ ИСПОЛЬЗУЕМЫХ КЛАССОВ	7
5. РАЗРАБОТКА СОБСТВЕННЫХ КЛАССОВ	7
5.1. VECTOR	7
5.2. DETAIL	8
5.3. DATABASE.....	9
5.4. <i>indialog</i>	10
5.5. MAINWINDOW	11
6. РАЗРАБОТКА ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ	12
7. ИНСТРУКЦИЯ ПОЛЬЗОВАТЕЛЮ	13
7.1. РАБОТА С ФАЙЛАМИ.....	13
7.1.1. СОЗДАНИЕ БАЗЫ	13
7.1.2. ОТКРЫТИЕ БАЗЫ ДАННЫХ.....	14
7.1.3. СОХРАНЕНИЕ БАЗЫ ДАННЫХ	15
7.1.4. ОБЪЕДИНЕНИЕ БАЗЫ	16
7.2. РАБОТА С ЗАПИСЯМИ	18
7.2.1. ДОБАВЛЕНИЕ ЗАПИСИ	18
7.2.2. УДАЛЕНИЕ ЗАПИСИ.....	19
7.2.3. РЕДАКТИРОВАНИЕ ЗАПИСИ	19
7.3. ДОПОЛНИТЕЛЬНЫЕ ВОЗМОЖНОСТИ	20
7.3.1. ПОИСК.....	20
7.3.2. О ПРОГРАММЕ.....	20
7.3.3. ВЫХОД ИЗ ПРОГРАММЫ	21
8. ЗАКЛЮЧЕНИЕ	22
9. СПИСОК ЛИТЕРАТУРЫ.....	23
10. ПРИЛОЖЕНИЕ	24
10.1. Файлы .cpp.....	24
main.cpp.....	24
Details.cpp	24

MainWindow.cpp.....	24
DataBase.cpp	29
indialog.cpp.....	39
main.h	32
indialog.h	32
Details.h	32
MainWindow.h	33
vector.h.....	34
DataBase.h	38

1. ПОСТАНОВКА ЗАДАЧИ

Создать базу данных, работающую на основе контейнера, который следует разработать в процессе выполнения курсовой работы. В качестве прототипа контейнера можно использовать контейнеры `vector` или `list` из библиотеки `STL`. Для работы с контейнером следует предусмотреть итератор. Остальные требования к контейнеру определяются студентом самостоятельно.

Разрабатываемая база данных должна быть создана использованием `C++` и библиотеки `Qt5`. Среда программирования `Qt Creator`.

В качестве элементов контейнеров должны использоваться объекты класса деталей (комплектующих) для компьютера (наименование, тип, марка, фирма изготовитель и т.д)

База данных должна обеспечивать выполнение следующих операций:

- создание базы данных
- объединение баз данных
- добавления и удаления записей
- просмотр базы данных
- редактирования записей
- поиск данных (найти данные, удовлетворяющие заданному критерию)
- запись на диск
- чтения с диска.

Интерфейсная часть программы должна содержать следующие компоненты:

- окно «ABOUT»
- главное меню
- всплывающее меню
- строку состояния
- панель инструментов
- подсказки по командам меню и панели инструментов.

2. ВЫБОР И ОБОСНОВАНИЕ ТИПА РАЗРАБАТЫВАЕМОГО КОНТЕЙНЕРА

В данной курсовой работе необходимо использовать контейнер для хранения объектов. Причем контейнер необходимо разработать самостоятельно, а не подключить, например, из стандартной библиотеки. В качестве прототипа контейнера можно использовать контейнеры `vector` или `list` из библиотеки STL. В качестве прототипа разрабатываемого контейнера я решила использовать контейнер `vector` из библиотеки STL. Выбор данного контейнера объясняется простотой своей разработки и организации, а также возможностью произвольного доступа к своим элементам, чего нет у контейнера типа `list`. Контейнер `list` обладает только последовательным доступом.

Разрабатываемый контейнер обладает всеми функциями, необходимыми для выполнения задачи:

- ✚ создание пустого контейнера; на основе массива; а также копии уже имеющегося контейнера того же типа;
- ✚ произвольный доступ к элементам;
- ✚ добавление элементов в конец занятого массива;
- ✚ удаление произвольного элемента массива;

Внутри класса контейнера определены ключевые слова `iterator`, `reference`, а также функции `begin()` и `end()`, в связи с чем возможно использование стандартного для C++11 цикла `for()`, используемого для контейнеров.

В данном примере показан проход по всему контейнеру и распечатка его значений:

```
Vector<int> vector = {1, 2, 3, 4, 10};  
for (auto i : vector) {  
    cout << i << " ";  
}
```

3. РАЗРАБОТКА СТРУКТУРЫ ПРОГРАММЫ

Программа должна состоять из графического интерфейса и кода, который остается «за кадром». Интерфейс составляют: элемент отображения таблицы TableView, строка поиска по таблице LineEdit, панель инструментов, главное меню и статусная строка. А также диалоговые окна: добавление записи в таблицу AddWindow, редактирование записи EditWindow и окно «О программе».

Фоновую часть составляют классы: контейнер Vector, содержимое контейнера – класс комплектующих Detail и сама база данных DataBase, представляющая собой таблицу, которая хранит в себе данные и содержит методы обработки базы.

Таким образом, программу составляют модули:

- ✚ main.cpp
- ✚ Details.cpp, Details.h
- ✚ vector.h
- ✚ DataBase.cpp, DataBase.h
- ✚ MainWindow.cpp, MainWindow.h, MainWindow.ui
- ✚ *indialog.cpp, indialog.h*

Подробнее о каждом модуле я расскажу позже.

4. ВЫБОР И ОБОСНОВАНИЕ ИСПОЛЬЗУЕМЫХ КЛАССОВ

Vector – класс-контейнер выбранного типа

MainWindow – класс для работы с интерфейсом программы

DataBase – класс с функциями, предназначенными для работы с базой данных. Этот класс связан с классами Vector и MainWindow.

Detail – класс, хранящий в себе поля выбранного типа элементов (комплектующие компьютера). Связан с классом DataBase отношением включения.

indialog – класс для добавления элемента в базу *и редактирования элемента базы.*

5. РАЗРАБОТКА СОБСТВЕННЫХ КЛАССОВ

5.1. VECTOR

Шаблонный класс, предназначенный для работы с элементами пользовательского типа.

Класс содержит поля:

size_type _capacity; // объем выделенной памяти

size_type _size; // количество элементов в контейнере

iterator _arr; // указатель на первый элемент контейнера

Для возможности использования итератора в разделе public объявлен тип `typedef T* iterator.`

Контейнер предусматривает следующие операции:

- `iterator begin ( )` и `iterator end ( )` – указатели на первый и следующий за последним элементы контейнера;
- `void append ( )` – добавляет элемент в конец занятой части контейнера;
- перегруженный оператор `[ ]` возвращает элемент контейнера, номер которого указан квадратных скобках;
- перегруженный оператор `=` задаёт значения в контейнере (присваивает объекту контейнера другой контейнер того же типа, проверяя перед этим объекты на равенство адресов памяти);
- перегруженный оператор `<<` соответствует операции `append()`, то есть добавляет элемент в конец занятой части контейнера;
- `bool isEmpty ( )` – возвращает `true`, если занятая часть контейнера пуста, иначе возвращает `false`;
- `void clear ( )` – удаляет все элементы из контейнера.
- `void remove (size_type index)` – удаляет элемент с индексом `index`;
- `void remove ( )` – удаляет последний элемент;
- `void reserve (size_type n)` – выделяет память под `n` элементов;
- `reference at (size_type index)` – операция доступа к элементу вектора по индексу. Возвращает ссылку на элемент по индексу `index`.

Дополнительно выполняется проверка на выход за границы диапазона вектора. В случае выхода индекса за пределы диапазона вектора генерирует исключение;

5.2. DETAIL

Класс, характеризующий детали компьютера

Содержит поля:

`QString _comp; // название детали`

```
QString _model; // модель  
QString _number; // артикул  
QString _country; // страна производства  
int _year; // год выпуска
```

В классе имеются методы доступа к представленным полям.

5.3. DATABASE

Класс, содержащий в себе базу данных и методы для работы с ней.

Содержит поля:

```
QString _filename; // имя открытого файла  
Vector<Detail> _details; // контейнер деталей  
Vector<QString> _headers; // шапка таблицы  
bool _isModified; // модифицирована ли база
```

Предусматривает переопределенные методы, наследованные от абстрактного класса табличной модели QAbstractTableModel:

- `int rowCount ( )` – возвращает количество строк таблицы;
- `int columnCount ( )` – возвращает количество столбцов таблицы;
- `QVariant data ( )` – метод, возвращающий то, что будет отображаться в ячейках таблицы;
- `bool setData ( )` – метод, вызываемый, когда пользователь изменил ячейку таблицы. Редактирует данные в контейнере `_details`;
- `QVariant headerData ( )` – метод, возвращающий то, что будет отображаться в шапке таблицы;
- `Qt::ItemFlags flags ( )` – устанавливает параметры для ячеек таблицы:
можно ли ячейки выделять, редактировать и т.д.;

Методы для работы с самой базой:

- `void loadFile (const QString& filename)` – загружает базу данных из файла, имя которого передано в параметрах;
- `void saveFile (const QString& filename)` – сохраняет открытую базу данных в файл, имя которого передано в параметрах;
- `void mergeFile (const QString& filename)` – добавляет базу данных, загружаемую из файла в конец текущей базы;
- `void addEntry (const QStringList& characteristics)` – добавляет запись в конец базы. В параметрах передается список характеристик для записи (деталь, модель, артикул, страна, год) в виде строк.
- `void editEntry (int row, const QStringList& characteristics)` – редактирует элемент таблицы с указанным индексом, заменяет его поля на те, что переданы в параметрах;
- `void removeEntry (int row)` – удаляет элемент с указанным индексом из таблицы;
- Методы доступа `QString filename()` и `bool isModified()`, возвращающие значения соответствующих полей;

5.4. `indialog`

Класс диалогового окна, предназначенный для ввода значений полей и дальнейшей их передачи в функцию, обеспечивающую добавление нового элемента с введенными полями в базу данных.

Имеет методы, возвращающие значения вводимых полей.

Имеет конструктор с параметрами, принимающий значения полей от выделенной строки таблицы и устанавливающий текст элементов ввода равным этим значениям.