ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ "САНКТ-ПЕТЕРБУРГСКИЙ

ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М. А. БОНЧ-БРУЕВИЧА"

Факультет инфокоммуникационных сетей и систем Кафедра программной инженерии и вычислительной техники

КУРСОВАЯ РАБОТА

по дисциплине «Объектно-ориентированное программирование»

ВАРИАНТ 24

Выполнил:

студент 2 курса

ИКПИ-14

Якубова О.А.

Проверил:

Старший преподаватель кафедры ПИиВТ

Петрова О.Б.

Санкт-Петербург

2022

**Оглавление**

[1. ПОСТАНОВКА ЗАДАЧИ 3](#_Toc122374102)

[2. ВЫБОР И ОБОСНОВАНИЕ ТИПА РАЗРАБАТЫВАЕМОГО КОНТЕЙНЕРА 4](#_Toc122374103)

[3. РАЗРАБОТКА СТРУКТУРЫ ПРОГРАММЫ 5](#_Toc122374104)

[4. ВЫБОР И ОБОСНОВАНИЕ ИСПОЛЬЗУЕМЫХ КЛАССОВ 6](#_Toc122374105)

[5. РАЗРАБОТКА СОБСТВЕННЫХ КЛАССОВ 6](#_Toc122374106)

[6. Разработка интерфейса пользователя 11](#_Toc122374107)

[7. Инструкция пользователя 12](#_Toc122374108)

[7.1. Работа с файлами 12](#_Toc122374109)

[7.1.1. Создание базы данных 12](#_Toc122374110)

[7.1.2. Открытие базы данных 12](#_Toc122374111)

[7.1.3. Сохранение базы данных 12](#_Toc122374112)

[7.1.4. Объединение двух баз данных 12](#_Toc122374113)

[7.2. Работа с записями 12](#_Toc122374114)

[7.2.1. Добавление записи 12](#_Toc122374115)

[7.2.2. Удаление записи 12](#_Toc122374116)

[7.2.3. Редактирование записи 12](#_Toc122374117)

[7.3. Дополнительные возможности 12](#_Toc122374118)

[7.3.1. Поиск 12](#_Toc122374119)

[7.3.2. О программе 12](#_Toc122374120)

[7.3.3. Выход из программы 12](#_Toc122374121)

[8. Заключение 13](#_Toc122374122)

[9. Список литературы 14](#_Toc122374123)

[10. Приложение 15](#_Toc122374124)

# 1. ПОСТАНОВКА ЗАДАЧИ

Создать базу данных, работающую на основе контейнера, который следует разработать в процессе выполнения курсовой работы. В качестве прототипа разрабатываемого 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 Qt6. Среда программирования Qt Creator.

В качестве элементов контейнеров, в соответствии с вариантом 24, должны использоваться объекты классов, характеризующих смартфон (производитель, функции и т.д.)

База данных должна обеспечивать выполнение следующих операций:

* + создание базы данных,
  + объединение баз данных,
  + добавления и удаления записей,
  + редактирования записей,
  + просмотра базы данных,
  + 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, разработанный самостоятельно.

В качестве прототипа разрабатываемого контейнера выбран list (динамический список), потому что он не требует предварительно выделенной памяти и перераспределения памяти для всего списка, если добавлен новый элемент, а выделенная память закончилась. Кроме того, добавление и удаление элементов, независимо от их места в списке, производиться за меньшую сложность именно в контейнере типа list, а прежние итераторы в таком случае остаются действительными.

Разрабатываемый контейнер должен обладать следующими функциями:

* Наличие итераторов begin и end;
* Создание пустого контейнера;
* Очистка контейнера
* Получение элемента по заданной позиции
* Добавление элемента в заданную позицию
* Удаление элемента по заданной позиции
* Изменение содержимого элемента по индексу

# 3. РАЗРАБОТКА СТРУКТУРЫ ПРОГРАММЫ

Программа состоит из интерфейса и кода. Интерфейс составляют: главное меню QMenuBar, панель инструментов QToolBar, элемент отображения таблицы QTableWidget, подпись QLabel, кнопка для перехода к следующей найденной строке QPushButton, строка поиска по таблице QLineEdit, статусная строка QStatusBar. А также диалоговые окна: добавление и редактирование записей AddData (6 текстовых подписей QLabel, характеристик смартфона: модель, производитель, цвет, размер дисплея, ёмкость батареи, год выпуска; 6 текстовых полей QLineEdit) и окно «О программе», содержащее информацию о выполняемом задании и авторе курсовой работы.

Код представляет собой несколько классов: контейнер List c вложенным в него классом Iterator, содержимое контейнера – класс, характеризующий смартфон, реализация работы окна AddData для добавления и редактирования записей в базе данных, функции MainWindow для работы с базой данных, чтения с диска и записи на диск, поиска по таблице и т.д.

Таким образом, программу составляют модули:

* main.cpp
* mainwindow.cpp; mainwindow.h; mainwindow.ui
* smartphone.cpp; smartphone.h
* list.cpp; list.h
* adddata.cpp; adddata.h; adddata.ui

# 4. ВЫБОР И ОБОСНОВАНИЕ ИСПОЛЬЗУЕМЫХ КЛАССОВ

List – класс-контейнер для хранения элементов базы данных.

Iterator – класс для реализации итераторов для контейнера List. Связан с классом List, так как является вложенным в него.

Smartphone – класс, характеризующий смартфоны. Связан с классом List отношением включения.

AddData – класс для добавления и редактирования записей базы данных, позволяет работать с интерфейсом окна.

MainWindow – класс для работы с интерфейсом программы. Позволяет выполнять необходимые операции с базами данных: создание базы данных, объединение баз данных, вызов окна для добавления и редактирования записей, удаление записей, просмотр базы данных в виде таблицы, поиск данных, запись на диск, чтения с диска.

# 5. РАЗРАБОТКА СОБСТВЕННЫХ КЛАССОВ

## 5.1. Smartphone

Класс, характеризующий смартфоны.

Содержит поля:

* QString developer; // производитель смартфона
* QString model; // название модели
* QString color; // цвет смартфона
* QString display; // размер дисплея, см
* QString battery; // ёмкость батареи
* QString year; // год выпуска
* Smartphone \*next; // указатель на следующий узел

Класс предусматривает следующие операции:

* перегруженный operator= присваивает соответствующим полям одного объекта контейнера значения полей другого контейнера того же типа;
* void setDeveloper (QString) устанавливает значение производителя смартфона в соответствующем поле класса;
* void setModel (QString) устанавливает значение модели смартфона в соответствующем поле класса;
* void setColor (QString) устанавливает значение цвета смартфона в соответствующем поле класса;
* void setDisplay (QString) устанавливает значение размера смартфона в соответствующем поле класса;
* void setBattery (QString) устанавливает значение ёмкости батареи смартфона в соответствующем поле класса;
* void setYear (QString) устанавливает значение года выпуска смартфона в соответствующем поле класса;
* QString getDeveloper () возвращает значение производителя смартфона типа QString;
* QString getModel () возвращает значение модели смартфона типа QString;
* QString getColor () возвращает значение цвета смартфона типа QString;
* QString getDisplay () возвращает значение размера дисплея смартфона типа QString;
* QString getBattery () возвращает значение ёмкости батареи смартфона типа QString;
* QString getYear () возвращает значение года выпуска смартфона типа QString;
* void setNext (Smartphone\*) установка указателя на следующий узел. Необходимо для реализации динамического списка;
* Smartphone\* getNext () получение указателя на следующий узел. Необходимо для реализации динамического списка;

## 5.2. List

Класс, предназначенный для работы с элементами пользовательского типа.

Содержит поля:

* Smartphone\* head; // указатель на верхний (первый) узел в списке
* int size; // размер списка

Для возможности использования итераторов внутри класса List был разработан класс Iterator.

Контейнер предусматривает следующие операции:

* Iterator begin() и Iterator end() – указатели на первый и следующий за последним элементы контейнера;
* void push\_front (QString developer\_, QString model\_, QString color\_, QString display\_, QString battery\_, QString year\_) – добавление нового узла в начало списка;
* void push\_back (QString developer\_, QString model\_, QString color\_, QString display\_, QString battery\_, QString year\_) – добавление нового узла в конец списка;
* bool push(int idx, QString developer\_, QString model\_, QString color\_, QString display\_, QString battery\_, QString year\_) – добавление нового узла в середину списка по индексу. В случае, если добавление прошло успешно, возвращает true, иначе – false;
* bool pop\_front(), bool pop\_back() и bool pop(int idx) – удаление элемента в начале, конце и середине списка по индексу соответственно. Если удаление прошло успешно, возвращает true, иначе – false;
* Smartphone\* get\_front(), Smartphone\* get\_back() и Smartphone\* get(int idx) – получение указателя на верхний узел, последний и по заданной позиции. Осуществляется проверка на попытку выйти за пределы контейнера.
* void clean() – очистка контейнера, освобождение памяти
* int getSize() – получение размера списка
* bool changeDeveloper(int idx, QString developer), bool changeModel(int idx, QString a), bool changeColor(int idx, QString a), bool changeDisplay(int idx, QString a), bool changeBattery(int idx, QString a) и bool changeYear(int idx, QString a) – изменение значений характеристик смартфона в узле по индексу с проверкой на выход за пределы списка;

## 5.3. Iterator

Класс итераторов для контейнера List.

Содержит поля:

* Smartphone\* cur – указатель на узел, на который указывает итератор;

Итератор предусматривает следующие операции:

* перегруженный operator+ перемещает итератор на заданное количество узлов вперёд;
* перегруженный operator++ перемещает итератор на один узел вперёд;
* перегруженные operator!= и operator< возвращают true, если узел, на который указывает первый итератор, не равен узлу, на который указывает второй, иначе false;
* перегруженные operator== возвращает true, если узел, на который указывает первый итератор, равен узлу, на который указывает второй, иначе false;
* QString getModel (), QString getDeveloper (), QString getColor (), QString getDisplay (), QString getBattery () и QString getYear () – получение характеристик смартфона, хранящихся в узле, на который указывает итератор;

## 5.4. AddData

Класс диалогового окна, предназначенный для ввода значений полей и дальнейшей их передачи в функции, обеспечивающие добавление нового элемента в базу данных или редактирование уже имеющегося.

Позволяет производить следующие операции:

* Smartphone getData() – возвращает узел с установленными пользователем значениями полей (характеристик смартфона)
* void setDeveloper(QString arg), void setModel(QString arg), void setColor(QString arg), void setDisplay(QString arg), void setBattery(QString arg), void setYear(QString arg) – позволяют устанавливать значения соответствующих характеристик смартфона в текстовых полях;
* реагирует на то, что выбрал пользователь в окне: «Ok» или «Cancel». Если пользователь выбирает «Ok», то введённые им изменения вступают в силу, иначе – нет, окно закрывается, база данных не меняется.

## 5.5. MainWindow

Класс для работы с графическим интерфейсом программы.

Содержит поля:

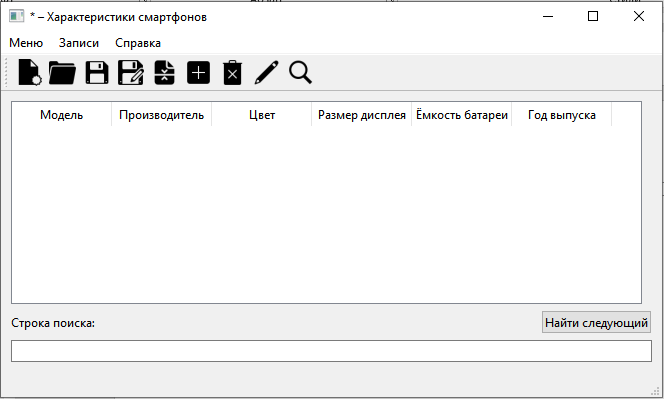
* Ui::MainWindow \*ui; // для обращения к элементам окна
* QString filepath = ""; // хранит путь к открытому файлу
* List dataBase; // база данных

Методы:

* void on\_aboutAct\_triggered() – показывает информационное окно с данными о программе и авторе;
* void on\_openItemAct\_triggered() – открытие базы данных из текстового файла;
* void on\_saveItemAct\_triggered() – сохранение базы данных, запись на диск
* void on\_addItemAct\_triggered() – добавление новой строки в базу данных, вызов окна AddData;
* void on\_createItemAct\_triggered() – создание новой базы данных, очистка таблицы в главном окне;
* void on\_exitAct\_triggered() – выход из приложения с предупреждением об утере несохранённых данных;
* void on\_saveAsItemAct\_triggered() – сохранение базы данных в другой файл;
* void on\_mergeAct\_triggered() – объединение одной базы данных с другой;
* void on\_deleteItemAct\_triggered() – удаление выделенной строки;
* void on\_editItemAct\_triggered() – редактирование выделенной строки, вызов окна AddData;
* void on\_findItemAct\_triggered() – поиск полей базы данных, идентичных введённому пользователем значению;
* void on\_pushButton\_2\_clicked() – показать следующую найденную строку
* void on\_lineEdit\_textChanged(const QString &arg1) – строка поиска
* void on\_tableWidget\_cellDoubleClicked(int row, int column) – при двойном нажатии мышкой на таблицу с базой данных снимает все выделения;

# 6. РАЗРАБОТКА ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ

На рисунке 1 изображено основное окно программы.



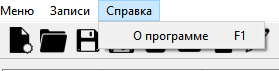
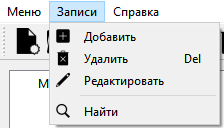
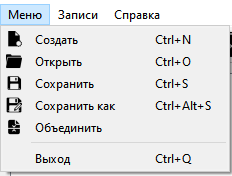
*Рисунок 1 – Основное окно программы*

В главном меню содержатся разделы «Меню», «Записи» и «Справка», их внешний вид представлен на рисунке 2.

Раздел «Меню» содержит операции создания, открытия, сохранения в текущий и новый файлы, объединения и выхода.

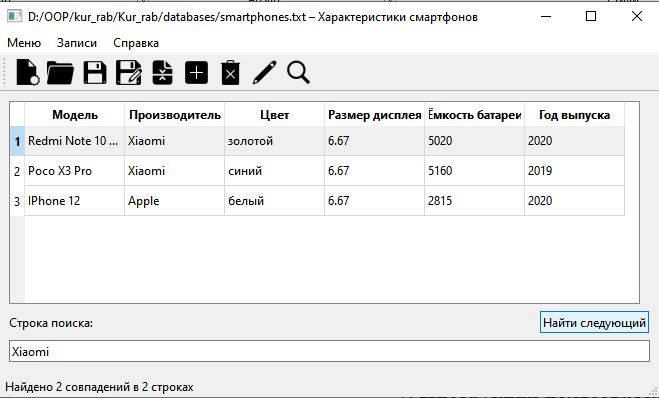
Раздел «Записи» содержит операции добавления, удаления, редактирования и поиска элементов базы данных.

Раздел «Справка» позволяет узнать информацию о программе.



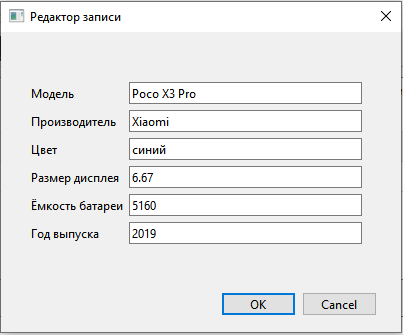
*Рисунок 2 – Содержимое главного меню*

Кроме главного меню, в программе есть панель инструментов, которая позволяет проводить те же операции. Строка поиска и кнопка «Найти следующее» позволяют находить количество совпадений в базе данных, её работа представлена на рисунке 3.



*Рисунок 3 – Поиск в базе данных*

При попытке изменить или добавить запись появляется окно, представленное на рисунке 4.



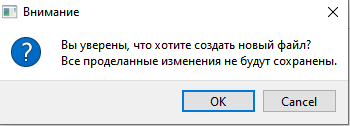
*Рисунок 4 – Окно редактора записи*

# 7. ИНСТРУКЦИЯ ПОЛЬЗОВАТЕЛЯ

## 7.1. Работа с файлами

### 7.1.1. Создание базы данных

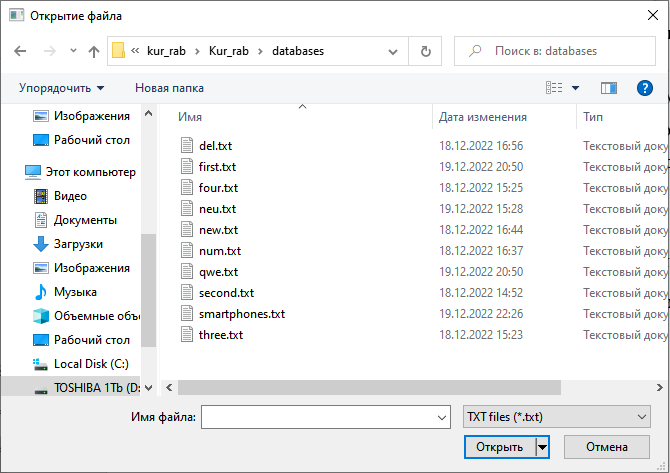
При запуске программы база данных создаётся автоматически. Если в процессе работы потребовалось создать новую базу данных, в главном меню выберите [Меню]->[Создать], или нажмите на значок , или используйте комбинацию клавиш Ctrl+N. Появится диалоговое окно, изображённое на рисунке 5 – выберите подходящий для вас вариант.



*Рисунок 5 – Диалоговое окно при создании новой базы данных*

### 7.1.2. Открытие базы данных

Чтобы открыть существующую базу данных, выберите [Меню] -> [Открыть], или нажмите на значок , или используйте комбинацию клавиш Ctrl+O. Появится диалоговое окно, изображённое на рисунке 5, если в текущую базу данных уже внесены изменения. Если в этом окне выбрать «Ok», появится окно для выбора файла, пример представлен на рисунке 6. Выбрать можно только текстовые файлы в формате «.txt».



*Рисунок 6 – Открытие файла*

### 7.1.3. Сохранение базы данных

Чтобы сохранить базу данных в текущий файл, выберите [Меню] -> [Сохранить], или нажмите на значок , или используйте комбинацию клавиш Ctrl+S. Если текущая база данных ещё не привязана к файлу, то будет открыто окно, как на рисунке 6, где можно создать файл для записи.

Если вы хотите сохранить базу данных в новом файле, выберите [Меню] -> [Сохранить как], или нажмите на значок , или используйте комбинацию клавиш Ctrl+S. Откроется окно, как на рисунке 6, где необходимо выбрать существующую базу данных или создать новый файл.

### 7.1.4. Объединение двух баз данных

Чтобы объединить текущую базу данных с другой, выберите [Меню] -> [Объединить], или нажмите на значок . Появится окно, похожее на то, что изображено на рисунке 6, где будет необходимо выбрать текстовый файл с базой данных, которую вы хотите вставить в конец своей базы данных.

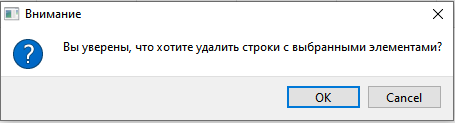
## 7.2. Работа с записями

### 7.2.1. Добавление записи

Чтобы добавить новую запись в конец таблицы, выберите [Записи] -> [Добавить] или нажмите на значок . Будет открыто окно, представленное на рисунке 4, в которое нужно записать значение полей для новой записи. Чтобы изменения вступили в силу, нажмите на кнопку «Ok».

### 7.2.2. Удаление записи

Чтобы удалить выделенную запись (или несколько записей), выберите [Записи] -> [Удалить], или нажмите на значок , или используйте клавишу Delete. Будет высвечено предупреждение, которое вы можете видеть на рисунке 7. Чтобы изменения вступили в силу, нажмите на кнопку «Ok».



*Рисунок 7 – Предупреждение при удалении записи*

### 7.2.3. Редактирование записи

Чтобы редактировать запись в таблице, выберите [Записи] -> [Редактировать] или нажмите на значок . Будет открыто окно, представленное на рисунке 4, в которое нужно записать значение полей для новой записи. Чтобы изменения вступили в силу, нажмите на кнопку «Ok».

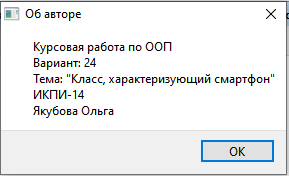
## 7.3. Дополнительные возможности

### 7.3.1. Поиск

Чтобы найти строку или строки, содержащее поле с заданным значением, введите это значение в строку поиска. Выберите [Записи] -> [Найти], или нажмите на значок , или на кнопку «Найти следующий». Пример работы поиска представлен на рисунке 3.

### 7.3.2. О программе

Чтобы узнать информацию о программе, выберите [Справка] -> [О программе] или нажмите F1. Появится окно, изображенное на рисунке 8.



*Рисунок 8 – Информация о программе*

### 7.3.3. Выход из программы

Для выхода из программы выберите [Меню] -> [Выход] или используйте сочетание клавиш Ctrl+Q. Появиться окно, предупреждающее вас, что данные не сохранятся.

# 8. Заключение

Поставленная задача выполнена успешно. Был разработан класс контейнера List для работы с базой данных, предусмотрены итераторы. Разработан класс Smartphone, характеризующий смартфоны. Были использованы разные виды отношений между классами, а именно, включение (Smartphone и List), вложение (List и Iterator), наследование (классы Qt). Данная программа выполняет основные операции над базой данных, определённые в постановке задачи. В ходе выполнения курсовой работы была освоена работа с основными возможностями языка C++, библиотеками Qt (QMainWindow, QString, QDialog, QDebug и др.) и редактором Qt Creator, с помощью которого был разработан интерфейс окон.

# 9. Список литературы

1. С.А. Коробов – Лекции по дисциплине «Объектно-ориентированное программирование», 2022 г.

2. Лафоре Р. – «Объектно-ориентированное программирование в C++», 2004 г.

3. Qt Documentation – <https://doc.qt.io> (дата последнего обращения 19.12.2022)

# 10. Приложение

## 10.1 Файлы .cpp

### 10.1.1. main.cpp

#include "mainwindow.h"

#include <QApplication>

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

{

QApplication a(argc, argv);

MainWindow w;

w.show();

return a.exec();

}

### 10.1.2. list.cpp

#include "list.h"

//СПИСОК

Smartphone\* List :: get\_front(){

if (head != NULL) return (head);

else return NULL;

}//получение значения верхнего узла

Smartphone\* List :: get\_back(){

return get(size-1);

}//получение значения верхнего узла

Smartphone\* List :: get(int idx){

if ((size != 0) || (head != NULL)) {

if (idx == 0){

return get\_front();

}

if (idx < size){

Smartphone\* ptr = head;

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

ptr = ptr->getNext();

if (ptr == NULL){

return NULL;

}

}

return ptr;

}

}

return NULL;

}//получение значения верхнего узла

List :: List():head(NULL), size(0){}//конструктор по умолчанию (создание пустого контейнера)

List :: List(QString developer\_, QString model\_, QString color\_, QString display\_, QString battery\_, QString year\_):head(new Smartphone(developer\_, model\_, color\_, display\_, battery\_, year\_, NULL)),size(1){}//конструктор с параметрами

List :: ~List(){

if (head != NULL){

Smartphone\* ptr = head;

while (ptr!=NULL){

head = ptr->getNext();

delete ptr;

ptr = head;

}

}

}//деструктор

List :: List(const List& A){

size = 0;

head = NULL;

Smartphone\* ptr = A.head;

for (int i = 0; (i < size) && (ptr != NULL); i++){

QString developer\_ = ptr->getDeveloper();

QString model\_ = ptr->getModel();

QString color\_ = ptr->getColor();

QString display\_ = ptr->getDisplay();

QString battery\_ = ptr->getBattery();

QString year\_ = ptr->getYear();

push\_front(developer\_, model\_, color\_, display\_, battery\_, year\_);

ptr = ptr->getNext();

}

}//конструкторо копирования

void List :: push\_front(QString developer\_, QString model\_, QString color\_, QString display\_, QString battery\_, QString year\_){

Smartphone\* A = new Smartphone(developer\_, model\_, color\_, display\_, battery\_, year\_, head);

head = A;

size++;

}//добавление нового узла в начало списка

void List :: push\_back(QString developer\_, QString model\_, QString color\_, QString display\_, QString battery\_, QString year\_){

if (head == NULL){

Smartphone\* A = new Smartphone(developer\_, model\_, color\_, display\_, battery\_, year\_, NULL);

head = A;

size++;

} else {

Smartphone\* ptr = head;

while (ptr->getNext() != NULL){

ptr = ptr->getNext();

}

Smartphone\* A = new Smartphone(developer\_, model\_, color\_, display\_, battery\_, year\_, NULL);

ptr->setNext(A);

size++;

}

}//добавление нового узла в конце списка

bool List :: push(int idx, QString developer\_, QString model\_, QString color\_, QString display\_, QString battery\_, QString year\_){

if (idx <= size){

if (idx <= 0) {

push\_front(developer\_, model\_, color\_, display\_, battery\_, year\_);

return true;

}

if (idx >= size) {

push\_back(developer\_, model\_, color\_, display\_, battery\_, year\_);

return true;

}

Smartphone\* ptr = head;

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

ptr = ptr->getNext();

if (ptr == NULL){

return false;

}

}

Smartphone\* A = new Smartphone(developer\_, model\_, color\_, display\_, battery\_, year\_, ptr->getNext());

ptr->setNext(A);

size++;

} else {

push\_back(developer\_, model\_, color\_, display\_, battery\_, year\_);

return true;

}

return false;

}//добавление нового узла в середину списка

bool List :: pop\_front(){

if ((size != 0) || (head != NULL)){

Smartphone\* A = head;

head = A->getNext();

delete A;

size--;

return true;

} else return false;

}//удаление узла из начала списка

bool List :: pop\_back(){

if ((size != 0) || (head != NULL)) {

size--;

Smartphone\* ptr = head;

while (ptr->getNext()->getNext() != NULL){

ptr = ptr->getNext();

}

delete ptr->getNext();

ptr->setNext(NULL);

return true;

} else return false;

}//удаление узла из конца списка

bool List :: pop(int idx){

if ((size != 0) || (head != NULL)) {

if (idx == 0){

return pop\_front();

}

if (idx == size-1){

return pop\_back();

}

if (idx < size){

Smartphone\* ptr = head;

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

ptr = ptr->getNext();

if (ptr == NULL){

return false;

}

}

Smartphone\* ptr\_next = ptr->getNext()->getNext();

delete ptr->getNext();

ptr->setNext(ptr\_next);

size--;

return true;

}

}

return false;

}//удаление узла из середины списка

void List :: clean(){

int n = size;

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

pop\_front();

}

return;

}//очистка списка

int List :: getSize(){

return size;

}//получение размера

bool List :: isEmpty(){

return (size == 0);

}//проверка на нулевой размер

bool List :: changeDeveloper(int idx, QString developer){

Smartphone\* ptr = get(idx);

if (ptr == NULL) return false;

ptr->setDeveloper(developer);

return true;

}//изменить элемент по индексу - производитель

bool List :: changeModel(int idx, QString a){

Smartphone\* ptr = get(idx);

if (ptr == NULL) return false;

ptr->setModel(a);

return true;

}//изменить элемент по индексу - модель

bool List :: changeColor(int idx, QString a){

Smartphone\* ptr = get(idx);

if (ptr == NULL) return false;

ptr->setColor(a);

return true;

}//изменить элемент по индексу - цвет

bool List :: changeDisplay(int idx, QString a){

Smartphone\* ptr = get(idx);

if (ptr == NULL) return false;

ptr->setDisplay(a);

return true;

}//изменить элемент по индексу - размер дисплея

bool List :: changeBattery(int idx, QString a){

Smartphone\* ptr = get(idx);

if (ptr == NULL) return false;

ptr->setBattery(a);

return true;

}//изменить элемент по индексу - ёмкость батареи

bool List :: changeYear(int idx, QString a){

Smartphone\* ptr = get(idx);

if (ptr == NULL) return false;

ptr->setYear(a);

return true;

}//изменить элемент по индексу – год

### 10.1.3. smartphone.cpp

#include "smartphone.h"

Smartphone :: Smartphone(){

developer = ""; //производитель

model = "";//название модели

color = "";//цвет

display = "";//размер дисплея, см

battery = "";//ёмкость батареи

year = "";//год выпуска

next = NULL;

}//конструктор по умолчанию

Smartphone :: Smartphone(QString de, QString m, QString c, QString di, QString b, QString y, Smartphone\* next\_){

developer = de; //производитель

model = m;//название модели

color = c;//цвет

display = di;//размер дисплея, см

battery = b;//ёмкость батареи

year = y;//год выпуска

next = next\_;

}//конструктор с параметрами

Smartphone :: Smartphone (const Smartphone &A){

developer = A.developer; //производитель

model = A.model;//название модели

color = A.color;//цвет

display = A.display;//размер дисплея, см

battery = A.battery;//ёмкость батареи

year = A.year;//год выпуска

next = A.next;

}//конструктор копирования

Smartphone :: ~Smartphone(){

next = NULL;

}//деструктор

//перегрузка оператора присваивания

Smartphone Smartphone :: operator= (Smartphone A){

developer = A.developer; //производитель

model = A.model;//название модели

color = A.color;//цвет

display = A.display;//размер дисплея, см

battery = A.battery;//ёмкость батареи

year = A.year;//год выпуска

//next = A.next;

return (\*this);

}

//оператор получения элемента по индексу

// Smartphone :: QString &operator[](int i){}

//очистка массива

// void clear();

//установка производителя

void Smartphone :: setDeveloper (QString de){

developer = de;

}

//получение производителя

QString Smartphone :: getDeveloper (){

return developer;

}

//установка модели

void Smartphone :: setModel (QString m){

model = m;

}

//получение модели

QString Smartphone :: getModel (){

return model;

}

//установка цвета

void Smartphone :: setColor (QString c){

color = c;

}

//получение цвета

QString Smartphone :: getColor (){

return color;

}

//установка размера дисплея

void Smartphone :: setDisplay (QString di){

display = di;

}

//получение размера дисплея

QString Smartphone :: getDisplay (){

return display;

}

//установка ёмкости батареи

void Smartphone :: setBattery (QString b){

battery = b;

}

//получение ёмкости батареи

QString Smartphone :: getBattery (){

return battery;

}

//установка года выпуска

void Smartphone :: setYear (QString y){

year = y;

}

//получение года выпуска

QString Smartphone :: getYear (){

return year;

}

void Smartphone :: setNext (Smartphone\* n){

next = n;

}

//получение указателя

Smartphone\* Smartphone :: getNext (){

return next;

}

### 10.1.4. mainwindow.cpp

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include <QMessageBox>

#include <QFileDialog>

#include <QFile>

#include <QDebug>

#include "list.h"

#include "smartphone.h"

#include "adddata.h"

MainWindow::MainWindow(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::MainWindow)

{

ui->setupUi(this);

setWindowTitle("\*" + filepath + " – Характеристики смартфонов");

ui->tableWidget->setColumnCount(6);

ui->tableWidget->setHorizontalHeaderLabels(QStringList() << "Модель" << "Производитель" << "Цвет" << "Размер дисплея" << "Ёмкость батареи" << "Год выпуска");

ui->tableWidget->setEditTriggers(QAbstractItemView::NoEditTriggers);

ui->tableWidget->setSelectionBehavior(QAbstractItemView::SelectRows);

ui->createItemAct->setToolTip("Сброс текущей таблицы");

}

MainWindow::~MainWindow()

{

delete ui;

}

void MainWindow::on\_aboutAct\_triggered() //О программе

{

QMessageBox::about(this, "Об авторе", "Курсовая работа по ООП\nВариант: 24\nТема: \"Класс, характеризующий смартфон\"\nИКПИ-14\nЯкубова Ольга");

}

void MainWindow::on\_openItemAct\_triggered() //Открыть

{

if (ui->tableWidget->rowCount() > 0 || filepath != "") {

QMessageBox msgBox;

msgBox.setWindowTitle("Внимание");

msgBox.setText("Вы уверены, что хотите выйти?\nВсе проделанные изменения не будут сохранены.");

msgBox.setIcon(QMessageBox::Question);

msgBox.setStandardButtons(QMessageBox::Ok | QMessageBox::Cancel);

msgBox.setDefaultButton(QMessageBox::Ok);

if (msgBox.exec() == QMessageBox::Cancel) return;

}

filepath = QFileDialog::getOpenFileName(this, "Открытие файла", "D://OOP/kur\_lab/Kur\_lab/databases/", "TXT files (\*.txt);;All files (\*.\*);;");

QFileInfo file(filepath);

if (file.exists() && file.isFile()) {

if (ui->tableWidget->rowCount() > 0 || filepath != "") {

ui->tableWidget->setRowCount(0);

dataBase.clean();

}

QFile file(filepath);

if (file.open(QIODevice::ReadOnly)) {

QTextStream in(&file);

while (!in.atEnd()) {

Smartphone item;

QString line = in.readLine();

QStringList fields = line.split(";");

if (fields.size() == 6) {

item.setDeveloper(fields[1].simplified());

item.setModel(fields[0].simplified());

item.setColor(fields[2].simplified());

item.setDisplay(fields[3].simplified());

item.setBattery(fields[4].simplified());

item.setYear(fields[5].simplified());

ui->tableWidget->setRowCount(ui->tableWidget->rowCount() + 1);

ui->tableWidget->setItem(ui->tableWidget->rowCount() - 1, 0, new QTableWidgetItem(item.getModel()));

ui->tableWidget->setItem(ui->tableWidget->rowCount() - 1, 1, new QTableWidgetItem(item.getDeveloper()));

ui->tableWidget->setItem(ui->tableWidget->rowCount() - 1, 2, new QTableWidgetItem(item.getColor()));

ui->tableWidget->setItem(ui->tableWidget->rowCount() - 1, 3, new QTableWidgetItem(item.getDisplay()));

ui->tableWidget->setItem(ui->tableWidget->rowCount() - 1, 4, new QTableWidgetItem(item.getBattery()));

ui->tableWidget->setItem(ui->tableWidget->rowCount() - 1, 5, new QTableWidgetItem(item.getYear()));

dataBase.push\_back(item.getDeveloper(), item.getModel(), item.getColor(), item.getDisplay(), item.getBattery(), item.getYear());

}

}

file.close();

ui->statusbar->showMessage("Записи из файла добавлены");

setWindowTitle(filepath + " – Характеристики смартфонов");

} else ui->statusbar->showMessage("Файл '" + filepath + "' не удалось открыть на чтение (" + file.errorString() + ")");

} else ui->statusbar->showMessage("Файл '" + filepath + "' не существует");

}

void MainWindow::on\_saveItemAct\_triggered() //Сохранить

{

if (filepath == "") {

MainWindow::on\_saveAsItemAct\_triggered();

return;

}

QFile file(filepath);

if (file.open(QIODevice::WriteOnly)) {

QTextStream out(&file);

int size = ui->tableWidget->rowCount();

if (size != dataBase.getSize()) QMessageBox::warning(this, "debug", "size != dataBase size");

for (int i = 0; (i < dataBase.getSize()) && (dataBase.get(i)!=NULL); i++){

QString model = dataBase.get(i)->getModel();

QString developer = dataBase.get(i)->getDeveloper();

QString color = dataBase.get(i)->getColor();

QString display = dataBase.get(i)->getDisplay();

QString battery = dataBase.get(i)->getBattery();

QString year = dataBase.get(i)->getYear();

out << model << ";" << developer << ";" << color << ";" << display << ";" << battery << ";" << year << "\n";

}

file.close();

ui->statusbar->showMessage("Файл записан");

setWindowTitle(filepath + " – Характеристики смартфонов");

}

}

void MainWindow::on\_createItemAct\_triggered() //Создать

{

if (ui->tableWidget->rowCount() > 0) {

QMessageBox msgBox;

msgBox.setWindowTitle("Внимание");

msgBox.setText("Вы уверены, что хотите создать новый файл?\nВсе проделанные изменения не будут сохранены.");

msgBox.setIcon(QMessageBox::Question);

msgBox.setStandardButtons(QMessageBox::Ok | QMessageBox::Cancel);

msgBox.setDefaultButton(QMessageBox::Ok);

if (msgBox.exec() == QMessageBox::Cancel) {

return;

}

}

ui->tableWidget->setRowCount(0);

dataBase.clean();

filepath = "";

setWindowTitle("\*" + filepath + " – Характеристики смартфонов");

}

void MainWindow::on\_exitAct\_triggered() //Выход

{

if (ui->tableWidget->rowCount() > 0 || filepath != "") {

QMessageBox msgBox;

msgBox.setWindowTitle("Внимание");

msgBox.setText("Вы уверены, что хотите выйти?\nВсе проделанные изменения не будут сохранены.");

msgBox.setIcon(QMessageBox::Question);

msgBox.setStandardButtons(QMessageBox::Ok | QMessageBox::Cancel);

msgBox.setDefaultButton(QMessageBox::Ok);

if (msgBox.exec() == QMessageBox::Cancel) {

return;

}

}

exit(0);

}

void MainWindow::on\_saveAsItemAct\_triggered() //Сохранить как

{

QString file\_name = QFileDialog::getSaveFileName(this, "Сохранить файл...", ".", "TXT files (\*.txt);;All files (\*.\*);;");

if (file\_name != "") {

filepath = file\_name;

MainWindow::on\_saveItemAct\_triggered();

}

}

void MainWindow::on\_mergeAct\_triggered() //Объединить

{

filepath = QFileDialog::getOpenFileName(this, "Открытие файла", "D://OOP/kur\_lab/Kur\_lab/databases/", "TXT files (\*.txt);;All files (\*.\*);;");

QFileInfo file(filepath);

if (file.exists() && file.isFile()) {

QFile file(filepath);

if (file.open(QIODevice::ReadOnly)) {

QTextStream in(&file);

while (!in.atEnd()) {

Smartphone item;

QString line = in.readLine();

QStringList fields = line.split(";");

if (fields.size() == 6) {

item.setDeveloper(fields[0].simplified());

item.setModel(fields[1].simplified());

item.setColor(fields[2].simplified());

item.setDisplay(fields[3].simplified());

item.setBattery(fields[4].simplified());

item.setYear(fields[5].simplified());

ui->tableWidget->setRowCount(ui->tableWidget->rowCount() + 1);

ui->tableWidget->setItem(ui->tableWidget->rowCount() - 1, 0, new QTableWidgetItem(item.getModel()));

ui->tableWidget->setItem(ui->tableWidget->rowCount() - 1, 1, new QTableWidgetItem(item.getDeveloper()));

ui->tableWidget->setItem(ui->tableWidget->rowCount() - 1, 2, new QTableWidgetItem(item.getColor()));

ui->tableWidget->setItem(ui->tableWidget->rowCount() - 1, 3, new QTableWidgetItem(item.getDisplay()));

ui->tableWidget->setItem(ui->tableWidget->rowCount() - 1, 4, new QTableWidgetItem(item.getBattery()));

ui->tableWidget->setItem(ui->tableWidget->rowCount() - 1, 5, new QTableWidgetItem(item.getYear()));

dataBase.push\_back(item.getDeveloper(), item.getModel(), item.getColor(), item.getDisplay(), item.getBattery(), item.getYear());

}

}

//qDebug () << dataBase.getSize() << "\n";

file.close();

ui->statusbar->showMessage("Записи из файла добавлены");

setWindowTitle(filepath + " – Характеристики смартфонов");

} else ui->statusbar->showMessage("Файл '" + filepath + "' не удалось открыть на чтение (" + file.errorString() + ")");

} else ui->statusbar->showMessage("Файл '" + filepath + "' не существует");

} //Объединение

void MainWindow::on\_addItemAct\_triggered() //Добавить

{

AddData myDialog;

if (myDialog.exec() == QDialog::Accepted){

Smartphone item = myDialog.getData();

ui->tableWidget->setRowCount(ui->tableWidget->rowCount() + 1);

ui->tableWidget->setItem(ui->tableWidget->rowCount() - 1, 0, new QTableWidgetItem(item.getModel()));

ui->tableWidget->setItem(ui->tableWidget->rowCount() - 1, 1, new QTableWidgetItem(item.getDeveloper()));

ui->tableWidget->setItem(ui->tableWidget->rowCount() - 1, 2, new QTableWidgetItem(item.getColor()));

ui->tableWidget->setItem(ui->tableWidget->rowCount() - 1, 3, new QTableWidgetItem(item.getDisplay()));

ui->tableWidget->setItem(ui->tableWidget->rowCount() - 1, 4, new QTableWidgetItem(item.getBattery()));

ui->tableWidget->setItem(ui->tableWidget->rowCount() - 1, 5, new QTableWidgetItem(item.getYear()));

dataBase.push\_back(item.getDeveloper(), item.getModel(), item.getColor(), item.getDisplay(), item.getBattery(), item.getYear());

ui->statusbar->showMessage("Запись добавлена");

setWindowTitle("\*" + filepath + " – Характеристики смартфонов");

}

} //Добавление элемента

void MainWindow::on\_deleteItemAct\_triggered()

{

auto list = ui->tableWidget->selectionModel()->selectedRows();

if (list.size() > 0) {

QMessageBox msgBox;

msgBox.setWindowTitle("Внимание");

msgBox.setText("Вы уверены, что хотите удалить строки с выбранными элементами?");

msgBox.setIcon(QMessageBox::Question);

msgBox.setStandardButtons(QMessageBox::Ok | QMessageBox::Cancel);

msgBox.setDefaultButton(QMessageBox::Ok);

if (msgBox.exec() == QMessageBox::Ok) {

for (auto k = list.rbegin(); k != list.rend(); ++k) {

auto idx = (\*k).row();

dataBase.pop(idx);

ui->tableWidget->removeRow(idx);

}

ui->statusbar->showMessage("Запись удалена");

setWindowTitle("\*" + filepath + " – Характеристики смартфонов");

}

} else ui->statusbar->showMessage("Ничего не выбрано");

} //Удаление элемента

void MainWindow::on\_editItemAct\_triggered()

{

auto list = ui->tableWidget->selectedItems();

if (list.size() > 0) {

AddData myDialog;

myDialog.setModel(list.at(0)->text());

myDialog.setDeveloper(list.at(1)->text());

myDialog.setColor(list.at(2)->text());

myDialog.setDisplay(list.at(3)->text());

myDialog.setBattery(list.at(4)->text());

myDialog.setYear(list.at(5)->text());

if (myDialog.exec() == QDialog::Accepted) {

Smartphone item = myDialog.getData();

list.at(0)->setText(item.getModel());

list.at(1)->setText(item.getDeveloper());

list.at(2)->setText(item.getColor());

list.at(3)->setText(item.getDisplay());

list.at(4)->setText(item.getBattery());

list.at(5)->setText(item.getYear());

dataBase.changeModel(list.at(0)->row(), item.getModel());

dataBase.changeDeveloper(list.at(1)->row(), item.getDeveloper());

dataBase.changeColor(list.at(2)->row(), item.getColor());

dataBase.changeDisplay(list.at(3)->row(), item.getDisplay());

dataBase.changeBattery(list.at(4)->row(), item.getBattery());

dataBase.changeYear(list.at(5)->row(), item.getYear());

ui->statusbar->showMessage("Запись изменена");

setWindowTitle("\*" + filepath + " – Характеристики смартфонов");

}

}

} //Редактирование элемента

int cur\_row = -1;

void MainWindow::on\_findItemAct\_triggered()

{

cur\_row = -1;

if (ui->tableWidget->rowCount() == 0) {

ui->statusbar->showMessage("Таблица пуста");

return;

}

QString text = ui->lineEdit->text();

if (text == "") {

ui->statusbar->showMessage("Строка поиска пуста");

return;

}

int count = 0;

int pr\_count = 0;

int rowcount = 0;

QString mas[6];

//ui->tableWidget->setSelectionMode(QAbstractItemView::SelectionMode::MultiSelection);

for (int i = 0; (i < dataBase.getSize()) && (dataBase.get(i)!=NULL); i++){

mas[0] = dataBase.get(i)->getModel();

mas[1] = dataBase.get(i)->getDeveloper();

mas[2] = dataBase.get(i)->getColor();

mas[3] = dataBase.get(i)->getDisplay();

mas[4] = dataBase.get(i)->getBattery();

mas[5] = dataBase.get(i)->getYear();

pr\_count = count;

for (int j = 0; j < 6; j++){

if (mas[j] == text) {

count++;

if (ui->tableWidget->rowCount() > i) ui->tableWidget->selectRow(i);

}

}

if (pr\_count != count) rowcount++;

}

QString textcount = QString::number(count);

QString textrowcount = QString::number(rowcount);

if (count%10 == 1 && count != 11) {

if (rowcount == 1) ui->statusbar->showMessage("Найдено " + textcount + " совпадение в 1 строкe");

else ui->statusbar->showMessage("Найдено " + textcount + " совпадение в " + textrowcount + " строках");

} else if (count%10 >= 2 && count%10 <= 4 && !(count > 11 && count < 15)) {

if (rowcount == 1) ui->statusbar->showMessage("Найдено " + textcount + " совпадения в 1 строкe");

else ui->statusbar->showMessage("Найдено " + textcount + " совпадения в " + textrowcount + " строках");

} else {

if (rowcount == 1) ui->statusbar->showMessage("Найдено " + textcount + " совпадений в 1 строкe");

else ui->statusbar->showMessage("Найдено " + textcount + " совпадений в " + textrowcount + " строках");

}

} //Поиск элемента

void MainWindow::on\_pushButton\_2\_clicked()

{

if (cur\_row > ui->tableWidget->rowCount()) cur\_row = -1;

if (ui->tableWidget->rowCount() == 0) {

ui->statusbar->showMessage("Таблица пуста");

return;

}

QString text = ui->lineEdit->text();

if (text == "") {

ui->statusbar->showMessage("Строка поиска пуста");

cur\_row = -1;

return;

}

int count = 0;

int pr\_count = 0;

int rowcount = 0;

QString mas[6];

int cur\_row\_pr = cur\_row;

int cur\_row\_first = -1;

for (int i = 0; (i < dataBase.getSize()) && (dataBase.get(i)!=NULL); i++){

mas[0] = dataBase.get(i)->getModel();

mas[1] = dataBase.get(i)->getDeveloper();

mas[2] = dataBase.get(i)->getColor();

mas[3] = dataBase.get(i)->getDisplay();

mas[4] = dataBase.get(i)->getBattery();

mas[5] = dataBase.get(i)->getYear();

pr\_count = count;

for (int j = 0; j < 6; j++){

if (mas[j] == text) {

count++;

if (cur\_row\_first == -1) cur\_row\_first = i;

if ((ui->tableWidget->rowCount() > i) && (cur\_row\_pr == cur\_row) && (cur\_row < i)) {

ui->tableWidget->selectRow(i);

cur\_row = i;

}

}

}

if (pr\_count != count) rowcount++;

}

if (cur\_row == cur\_row\_pr) {

cur\_row = cur\_row\_first;

ui->tableWidget->selectRow(cur\_row);

}

if (count == 0) {

ui->tableWidget->clearSelection();

}

QString textcount = QString::number(count);

QString textrowcount = QString::number(rowcount);

if (rowcount == 1) ui->statusbar->showMessage("Найдено " + textcount + " совпадений в 1 строкe");

ui->statusbar->showMessage("Найдено " + textcount + " совпадений в " + textrowcount + " строках");

//qDebug() << cur\_row;

} //Поиск: следующий

void MainWindow::on\_lineEdit\_textChanged(const QString &arg1)

{

cur\_row = -1;

} //Строка поиска изменилась

void MainWindow::on\_tableWidget\_cellDoubleClicked(int row, int column)

{

ui->tableWidget->clearSelection();

}

### 10.1.5. adddata.cpp

#include "adddata.h"

#include "ui\_adddata.h"

AddData::AddData(QWidget \*parent) :

QDialog(parent),

ui(new Ui::AddData)

{

ui->setupUi(this);

setWindowTitle("Редактор записи");

}

AddData::~AddData()

{

delete ui;

}

Smartphone AddData :: getData(){

Smartphone item;

item.setModel(ui->lineEdit->text().simplified().replace(";", ","));

item.setDeveloper(ui->lineEdit\_2->text().simplified().replace(";", ","));

item.setColor(ui->lineEdit\_3->text().simplified().replace(";", ","));

item.setDisplay(ui->lineEdit\_4->text().simplified().replace(";", ","));

item.setBattery(ui->lineEdit\_5->text().simplified().replace(";", ","));

item.setYear(ui->lineEdit\_6->text().simplified().replace(";", ","));

return item;

}

void AddData :: setDeveloper(QString arg){

ui->lineEdit\_2->setText(arg);

}

void AddData :: setModel(QString arg){

ui->lineEdit->setText(arg);

}

void AddData :: setColor(QString arg){

ui->lineEdit\_3->setText(arg);

}

void AddData :: setDisplay(QString arg){

ui->lineEdit\_4->setText(arg);

}

void AddData :: setBattery(QString arg){

ui->lineEdit\_5->setText(arg);

}

void AddData :: setYear(QString arg){

ui->lineEdit\_6->setText(arg);

}

void AddData::on\_buttonBox\_accepted()

{

accept();

}

void AddData::on\_buttonBox\_rejected()

{

reject();

}

## 10.2 Файлы .h

### 10.2.1. list.h

#ifndef LIST\_H

#define LIST\_H

#include "smartphone.h"

#include <cstddef>

#include <QString>

//список

class List {

private:

Smartphone\* head;//указатель на верхний (первый) узел в списке

int size;//размер списка

protected:

public:

friend class Iterator;

List();//конструктор по умолчанию (создание пустого контейнера)

List(QString developer\_, QString model\_, QString color\_, QString display\_, QString battery\_, QString year\_);//конструктор с параметрами

~List();//деструктор

List(const List& A);//конструкторо копирования

void push\_front(QString developer\_, QString model\_, QString color\_, QString display\_, QString battery\_, QString year\_);//добавление нового узла в начало списка

void push\_back(QString developer\_, QString model\_, QString color\_, QString display\_, QString battery\_, QString year\_);//добавление нового узла в конце списка

bool push(int idx, QString developer\_, QString model\_, QString color\_, QString display\_, QString battery\_, QString year\_);//добавление нового узла в середину списка

bool pop\_front();//удаление узла из начала списка

bool pop\_back();//удаление узла из конца списка

bool pop(int idx);//удаление узла из середины списка

Smartphone\* get\_front();//получение значения верхнего узла

Smartphone\* get\_back();//получение значения верхнего узла

Smartphone\* get(int idx);//получение значения верхнего узла

void clean();//очистка списка

int getSize();//получение размера

bool isEmpty();//проверка на нулевой размер

bool changeDeveloper(int idx, QString developer);//изменить элемент по индексу - производитель

bool changeModel(int idx, QString a);//изменить элемент по индексу - модель

bool changeColor(int idx, QString a);//изменить элемент по индексу - цвет

bool changeDisplay(int idx, QString a);//изменить элемент по индексу - размер дисплея

bool changeBattery(int idx, QString a);//изменить элемент по индексу - ёмкость батареи

bool changeYear(int idx, QString a);//изменить элемент по индексу - год

//ИТЕРАТОР ДЛЯ СПИСКА

class Iterator;

Iterator begin() {

return Iterator(head);

}//итератор начала

Iterator end() {

return Iterator(NULL);

}//итератор конца

class Iterator{

friend class List;

private:

Smartphone\* cur;

public:

Iterator(Smartphone\* node):cur(node){}

Iterator& operator+ (int n){

for (int i = 0; (cur != NULL) && (i < n); i++){

cur = cur->getNext();

}

return \*this;

}

Iterator& operator++ (){

if (cur != NULL){

cur = cur->getNext();

return \*this;

}

return \*this;

}

Iterator& operator++ (int){

if (cur != NULL){

cur = cur->getNext();

return \*this;

}

return \*this;

}

bool operator!= (const Iterator& it){

return cur != it.cur;

}

bool operator< (const Iterator& it){

return cur != it.cur;

}

bool operator== (const Iterator& it){

return cur == it.cur;

}

QString getModel (){

return cur->getModel();

}

QString getDeveloper (){

return cur->getDeveloper();

}

QString getColor (){

return cur->getColor();

}

QString getDisplay (){

return cur->getDisplay();

}

QString getBattery (){

return cur->getBattery();

}

QString getYear (){

return cur->getYear();

}

};

};

#endif // LIST\_H

### 10.2.2. mainwindow.h

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include "list.h"

#include <QMainWindow>

QT\_BEGIN\_NAMESPACE

namespace Ui { class MainWindow; }

QT\_END\_NAMESPACE

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

MainWindow(QWidget \*parent = nullptr);

~MainWindow();

private slots:

void on\_aboutAct\_triggered();

void on\_openItemAct\_triggered();

void on\_saveItemAct\_triggered();

void on\_addItemAct\_triggered();

void on\_createItemAct\_triggered();

void on\_exitAct\_triggered();

void on\_saveAsItemAct\_triggered();

void on\_mergeAct\_triggered();

void on\_deleteItemAct\_triggered();

void on\_editItemAct\_triggered();

void on\_findItemAct\_triggered();

void on\_pushButton\_2\_clicked();

void on\_lineEdit\_textChanged(const QString &arg1);

void on\_tableWidget\_cellDoubleClicked(int row, int column);

private:

Ui::MainWindow \*ui;

QString filepath = "";

List dataBase;

};

#endif // MAINWINDOW\_H

### 10.2.3. smartphone.h

#ifndef SMARTPHONE\_H

#define SMARTPHONE\_H

#include <qstring.h>

class Smartphone

{

public:

Smartphone();//конструктор по умолчанию

Smartphone(QString developer, QString model, QString color, QString display, QString battery, QString year, Smartphone\* next);//конструктор с параметрами

Smartphone (const Smartphone &A);//конструктор копирования

virtual ~Smartphone();//деструктор

//перегрузка оператора присваивания

Smartphone operator= (Smartphone A);

//установка производителя

void setDeveloper (QString);

//получение производителя

QString getDeveloper ();

//установка модели

void setModel (QString);

//получение модели

QString getModel ();

//установка цвета

void setColor (QString);

//получение цвета

QString getColor ();

//установка размера дисплея

void setDisplay (QString);

//получение размера дисплея

QString getDisplay ();

//установка ёмкости батареи

void setBattery (QString);

//получение ёмкости батареи

QString getBattery ();

//установка года выпуска

void setYear (QString);

//получение года выпуска

QString getYear ();

//установка указателя

void setNext (Smartphone\*);

//получение указателя

Smartphone\* getNext ();

protected:

private:

QString developer; //производитель

QString model;//название модели

QString color;//цвет

QString display;//размер дисплея, см

QString battery;//ёмкость батареи

QString year;//год выпуска

Smartphone \*next;//указатель на следующий узел

};

#endif // SMARTPHONE\_H

### 10.2.4. adddata.h

#ifndef ADDDATA\_H

#define ADDDATA\_H

#include <QDialog>

#include "smartphone.h"

namespace Ui {

class AddData;

}

class AddData : public QDialog

{

Q\_OBJECT

public:

explicit AddData(QWidget \*parent = nullptr);

~AddData();

Smartphone getData();

void setDeveloper(QString arg);

void setModel(QString arg);

void setColor(QString arg);

void setDisplay(QString arg);

void setBattery(QString arg);

void setYear(QString arg);

private slots:

void on\_buttonBox\_accepted();

void on\_buttonBox\_rejected();

private:

Ui::AddData \*ui;

};

#endif // ADDDATA\_H