

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет информационной безопасности

Кафедра инфокоммуникационных технологий

**ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ
ИНФОКОММУНИКАЦИОННЫХ СИСТЕМ
Часть 1**

**Лабораторная работа 2
Использование конструктора и деструктора.
Работа с динамической памятью.
Основы работы с GUI**



Минск 2022

Содержание

Лабораторная работа 2	
Использование конструктора и деструктора. Работа с динамической памятью. Основы работы с GUI.....	3
Конструкторы и деструкторы	3
Динамическое выделение памяти под объекты	5
Основы работы с GUI VS	6
Основы работы с GUI QT Creator.....	7
Задание к лабораторной работе 2.....	10

Лабораторная работа 2

Использование конструктора и деструктора.

Работа с динамической памятью.

Основы работы с GUI

Цель работы: Изучить основные способы создания конструкторов с захватом динамической памяти и деструкторов для ее освобождения. Ознакомится с основными Qt C++ классами.

Конструкторы и деструкторы

В C++ существуют два способа определить переменную. Можно сначала объявить переменную, а затем присвоить ей значение.

```
int number; // Объявление целочисленной переменной
number = 2; // Присвоение переменной значения
```

Или, определив переменную, сразу же инициализировать ее.

```
int number = 2; // Определение переменной и ее инициализация
```

Инициализация объединяет определение переменной с присвоением ей исходного значения, что ничуть не запрещает изменить это значение впоследствии. Инициализация гарантирует, что переменная никогда не останется без значения.

В рассмотренных ранее примерах поля объектов класса устанавливались "вручную" с помощью следующих инструкций.

```
product2.price = 145600; product2.avbegin = 40;
product2.avend = 5;
```

В профессионально написанных программах такой подход применяется редко. Для инициализации полей используется специальный метод, называемый *конструктором*.

Конструктор

Конструктор – это метод класса, который служит для инициализации объекта при его создании. Имя конструктора совпадает с именем самого класса. При необходимости конструктор может получать параметры, но не может возвращать значения, даже типа `void`. Конструктор без параметров называется стандартным.

```
имя_класса()
{
    операторы конструктора
}
```

Если реализация конструктора определяется за пределами класса, то в классе указывается прототип конструктора в следующем формате.

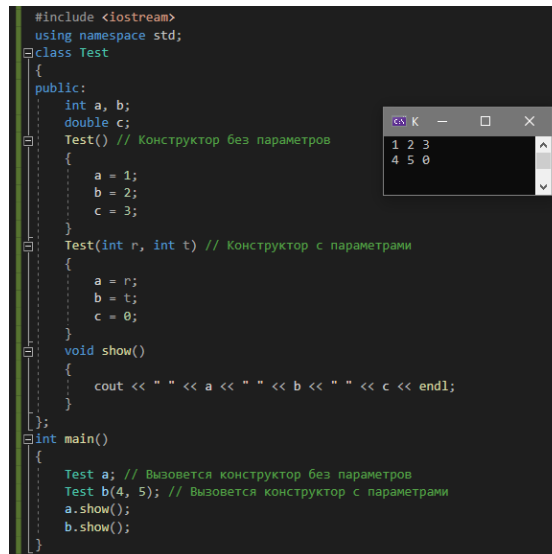
```
имя_класса();
// реализация за пределами класса
имя_класса::имя_класса()
{
    операторы конструктора
}
```

Обычно конструктор используется для придания переменным экземпляра, определенным в классе, начальных значений, а также для выполнения других начальных процедур, требуемых для создания полностью сформированного объекта.

Параметрические конструкторы (конструкторы с параметрами)

Параметры добавляются в конструктор так же, как они добавляются в метод: необходимо объявить их внутри круглых скобок после имени конструктора.

имя_класса() имя_объекта(список параметров);



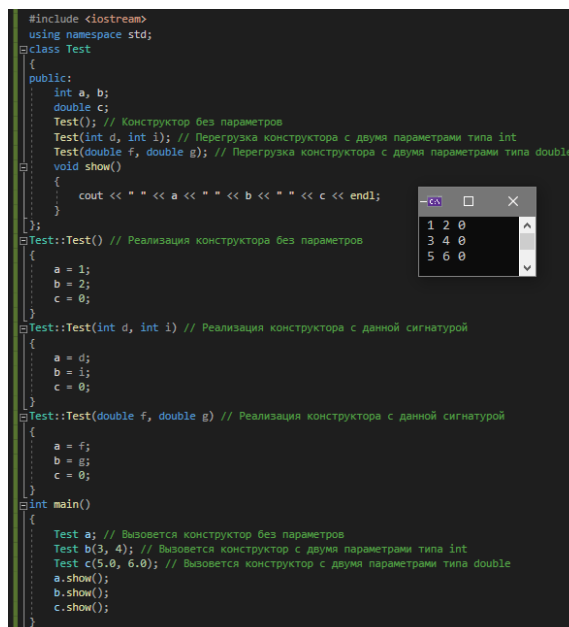
```
#include <iostream>
using namespace std;
class Test
{
public:
    int a, b;
    double c;
    Test() // Конструктор без параметров
    {
        a = 1;
        b = 2;
        c = 3;
    }
    Test(int r, int t) // Конструктор с параметрами
    {
        a = r;
        b = t;
        c = 0;
    }
    void show()
    {
        cout << " " << a << " " << b << " " << c << endl;
    }
};
int main()
{
    Test a; // Вызовется конструктор без параметров
    Test b(4, 5); // Вызовется конструктор с параметрами
    a.show();
    b.show();
}
```

Перегрузка конструкторов

Как и обычные функции, методы классов можно перегружать. В этом случае создается несколько вариантов одного и того же метода, но с разными прототипами. Отличие может быть связано с разным типом и количеством параметров (сигнатурой). Формально перегруженные методы могут быть (с точки зрения функциональности) абсолютно разными. Но хорошим стилем считается, если разные варианты перегруженного метода объединены общей идеей. Такой подход позволяет использовать единый интерфейс и при этом учесть особенности вызова соответствующего метода с разным набором аргументов.

Данная концепция получила название *полиморфизма*, одного из трех фундаментальных механизмов, на которых базируется объектно-ориентированное программирование.

Конструкторы также можно перегружать. Для перегрузки конструктора класса необходимо объявить разные его формы.



```
#include <iostream>
using namespace std;
class Test
{
public:
    int a, b;
    double c;
    Test(); // Конструктор без параметров
    Test(int d, int i); // Перегрузка конструктора с двумя параметрами типа int
    Test(double f, double g); // Перегрузка конструктора с двумя параметрами типа double
    void show()
    {
        cout << " " << a << " " << b << " " << c << endl;
    }
};
Test::Test() // Реализация конструктора без параметров
{
    a = 1;
    b = 2;
    c = 3;
}
Test::Test(int d, int i) // Реализация конструктора с данной сигнатурой
{
    a = d;
    b = i;
    c = 0;
}
Test::Test(double f, double g) // Реализация конструктора с данной сигнатурой
{
    a = f;
    b = g;
    c = 0;
}
int main()
{
    Test a; // Вызовется конструктор без параметров
    Test b(3, 4); // Вызовется конструктор с двумя параметрами типа int
    Test c(5.0, 6.0); // Вызовется конструктор с двумя параметрами типа double
    a.show();
    b.show();
    c.show();
}
```

Деструктор

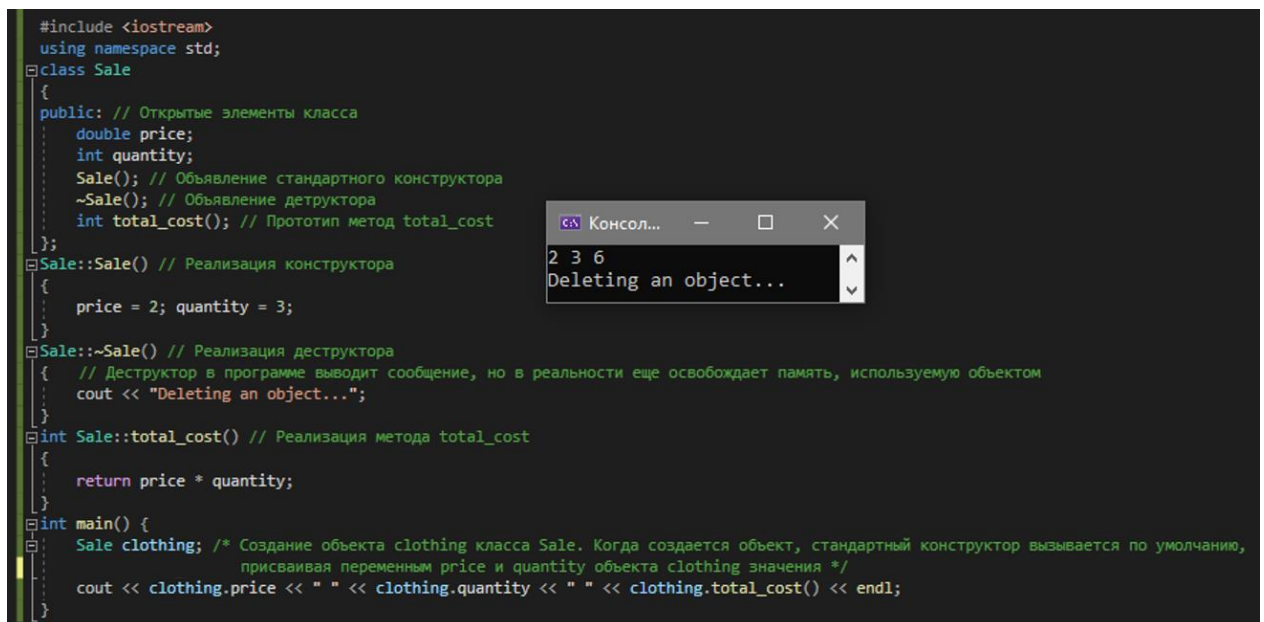
В некотором роде противоположностью конструктору является *деструктор*. Во многих случаях объекту при его удалении требуется выполнение некоторого действия или последовательности действий. Локальные объекты создаются при входе в свой блок и уничтожаются при выходе из блока. Глобальные объекты уничтожаются, когда завершается программа. Может быть много причин, приводящих к необходимости иметь деструктор. Например, объекту может понадобиться освободить ранее выделенную память или закрыть открытый им файл. В C++ такого рода операции выполняет *деструктор*.

Деструктор удаляет из памяти отработавшие объекты и освобождает выделенную для них память. Чтобы придать классу законченность, при объявлении конструктора необходимо объявить и деструктор, даже если ему нечего делать. Деструктору всегда присваивается имя класса с символом тильды (~) в начале. Деструкторы не получают аргументов и не возвращают значений!

Объявление деструктора класса будет выглядеть следующим образом:

```
~имя_класса();
```

Если конструктор или деструктор не созданы явно, то компилятор создаст их сам. Созданный компилятором конструктор будет стандартным, то есть без параметров. Конструкторы и деструкторы, созданные компилятором, не только не имеют аргументов, но и ничего не делают.



```
#include <iostream>
using namespace std;
class Sale
{
public: // Открытые элементы класса
    double price;
    int quantity;
    Sale(); // Объявление стандартного конструктора
    ~Sale(); // Объявление деструктора
    int total_cost(); // Прототип метода total_cost
};
Sale::Sale() // Реализация конструктора
{
    price = 2; quantity = 3;
}
Sale::~Sale() // Реализация деструктора
{
    // Деструктор в программе выводит сообщение, но в реальности еще освобождает память, используемую объектом
    cout << "Deleting an object...";
}
int Sale::total_cost() // Реализация метода total_cost
{
    return price * quantity;
}
int main() {
    Sale clothing; /* Создание объекта clothing класса Sale. Когда создается объект, стандартный конструктор вызывается по умолчанию,
                    присваивая переменным price и quantity объекта clothing значения */
    cout << clothing.price << " " << clothing.quantity << " " << clothing.total_cost() << endl;
}
```

Консольный вывод:

```
2 3 6
Deleting an object...
```

Динамическое выделение памяти под объекты

Для динамического выделения памяти под объекты класса используется оператор **new**. Сначала объявляется указатель на объект соответствующего класса, после чего под объект выделяется место в памяти и адрес передается указателю.

```
имя_класса *имя_указателя;
имя_указателя = new имя_класса;
```

Если при создании объекта конструктору необходимо передать аргументы, они указываются в круглых скобках после имени класса.

```
имя_класса *имя_указателя;
имя_указателя = new имя_класса(аргумент_1, аргумент_2);
```

Оператор **new** автоматически определяет размер выделяемой памяти для указанного типа, а также позволяет инициализировать вновь создаваемый

объект. Можно сказать, что оператор `new` устанавливает указатель на участок свободной динамической оперативной памяти.

Если выделить память не удастся, то возвращается нулевой указатель, что дает возможность проконтролировать процесс захвата участка динамической памяти.

Для удаления объекта из памяти используется оператор `delete`, после которого следует указатель на удаляемый объект.

```
delete имя_указателя;
```

Объект существует до тех пор, пока память не будет освобождена при помощи оператора `delete` (или до окончания работы программы).

В программе, представленной ниже продемонстрирована работа с динамической памятью. С помощью метода `show_cr` происходит увеличение значений полей класса в три раза.

```
#include <iostream>
#include <cstring>
#include <conio.h>
using namespace std;

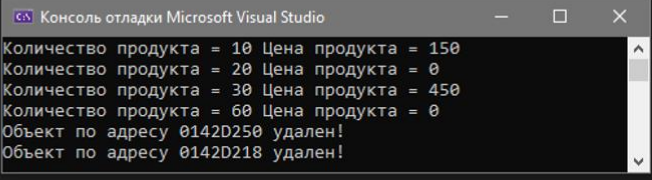
class Product // Объявление класса Product
{
public: // Открытые элементы класса
    int* quantity; // Количество продукта
    double* price; // Цена продукта
    Product(); // Стандартный конструктор
    Product(int a_quantity, double b_price); // Конструктор с параметрами
    ~Product(); // Деструктор
    void show_product() // Вывод информации о продукте
    {
        cout << "Количество продукта = " << *quantity << " Цена продукта = " << *price << endl;
    }
    void show_cr() // Вывод новой информации о продукте
    {
        *quantity *= 3;
        *price *= 3;
    }
};

Product::Product() // Стандартный конструктор
{
    quantity = new int(10); // Динамическое выделение памяти для переменных
    price = new double(150);
}

Product::Product(int a_quantity, double b_price) // Конструктор с параметрами
{
    quantity = new int(a_quantity);
    price = new double(b_price);
}

Product::~Product() // Деструктор
{
    cout << "Объект по адресу " << price << " удален! " << endl;
    delete(quantity, price); // Освобождение памяти
}

int main() {
    setlocale(LC_ALL, "Russian");
    Product product; // Создание объекта product класса Product. Вызовется стандартный конструктор
    Product product1(20, 300); // Создание объекта product1 класса Product. Вызовется конструктор с параметрами
    product.show_product();
    product1.show_product();
    product.show_cr();
    product1.show_cr();
    product.show_product();
    product1.show_product();
}
```



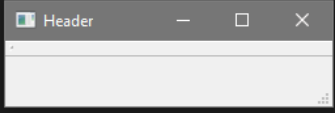
Основы работы с GUI VS

Стандартное окно

Следующая программа отображает на экране стандартное окно.

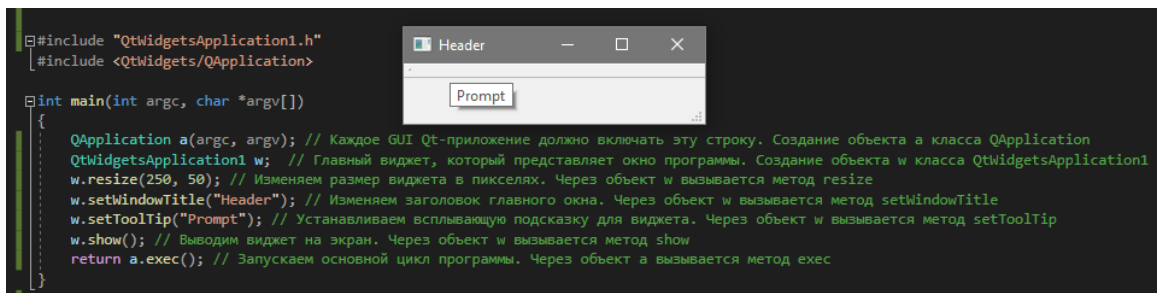
```
#include "QtWidgets/Application1.h"
#include <QtWidgets/QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv); // Каждое GUI Qt-приложение должно включать эту строку. Создание объекта a класса QApplication
    QWidgetApplication1 w; // Главный виджет, который представляет окно программы. Создание объекта w класса QWidgetApplication1
    w.resize(250, 50); // Изменяем размер виджета в пикселях. Через объект w вызывается метод resize
    w.setWindowTitle("Header"); // Изменяем заголовок главного окна. Через объект w вызывается метод setWindowTitle
    w.show(); // Выводим виджет на экран. Через объект w вызывается метод show
    return a.exec(); // Запускаем основной цикл программы. Через объект a вызывается метод exec
}
```



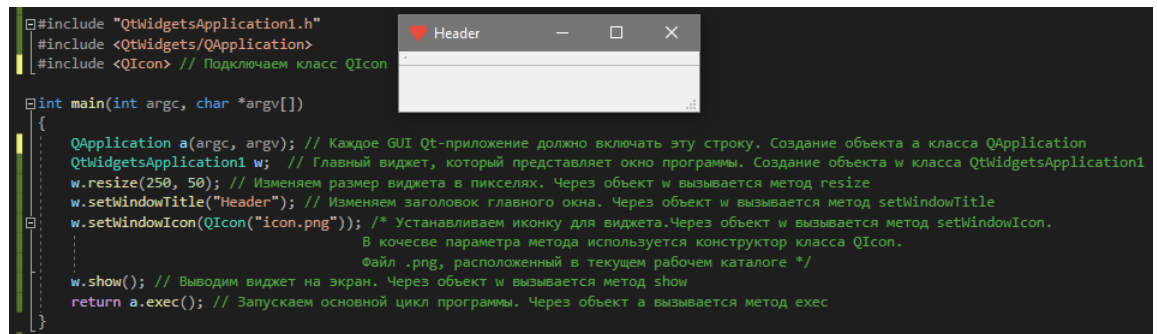
Всплывающая подсказка

В следующей программе демонстрируется создание всплывающей подсказки, которая отображается при наведении курсора на какой-нибудь элемент в приложении.



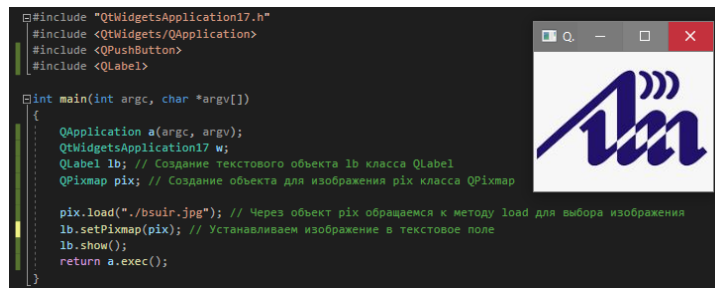
Иконка приложения

Большинство оконных менеджеров отображают этот значок в левом углу заголовка и на панели задач.



Вставка изображения

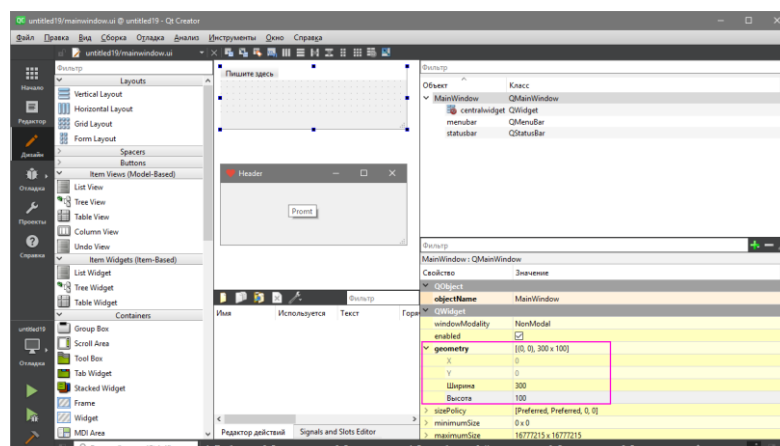
В фреймворке Qt нет компонента, который отвечает конкретно за изображения. Чтобы добавить изображение в программу необходимо добавить текстовый объект класса `QLabel` и при помощи класса `QPixmap` разместить в нем изображение. Все изображения лучше всего хранить в самом проекте.



Основы работы с GUI QT Creator

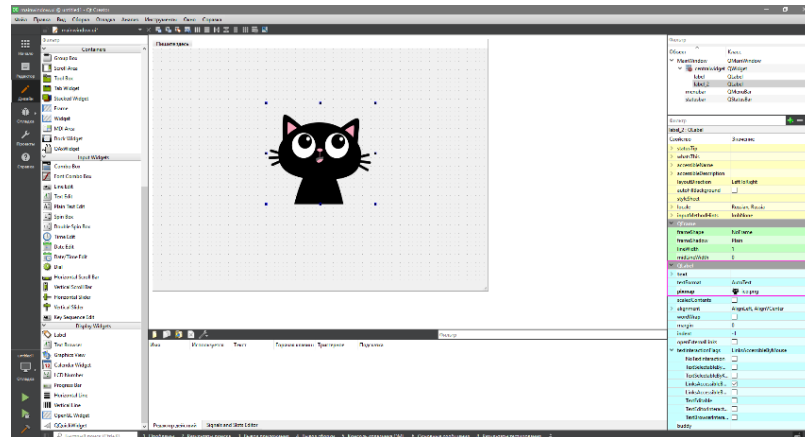
Стандартное окно, всплывающая подсказка, иконка приложения

Для установки названия, размеров, всплывающей подсказки и иконки приложения можно воспользоваться свойствами `geometry`, `windowTitle`, `tooltip`, `windowIcon` в режиме Дизайн, [смотреть](#).

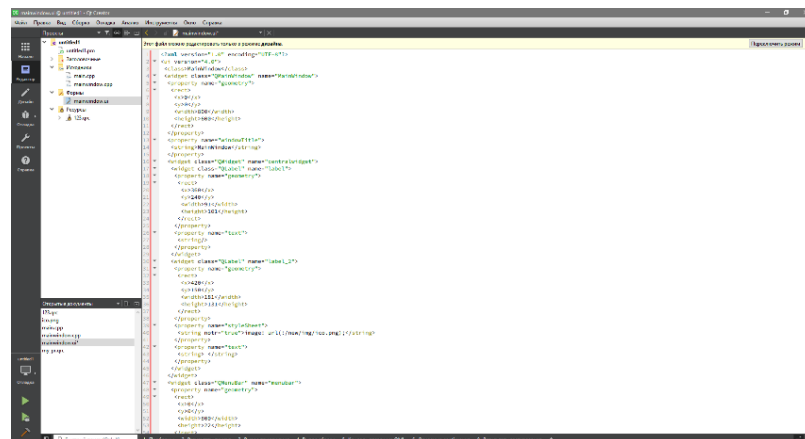


Вставка изображения

Как уже говорилось ранее, чтобы добавить изображение в программу необходимо добавить текстовый объект класса `QLabel` и при помощи класса `QPixmap` разместить в нем изображение, [смотреть](#).



Для того чтобы менять изображение согласно размеру объекта `QLabel` необходимо использовать ресурсы и каскадную таблицу стилей, [смотреть](#).

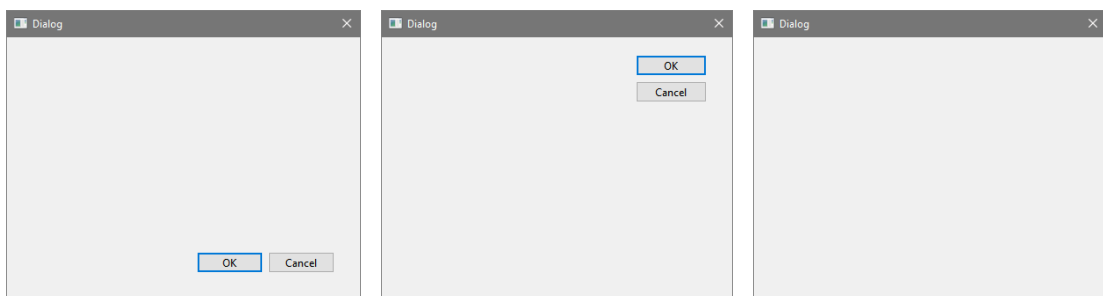


Создание окон

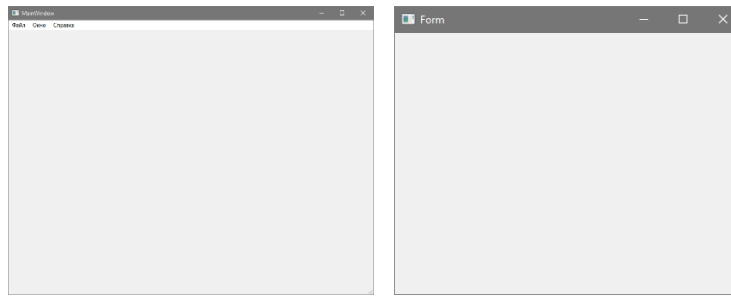
Шаблоны формы могут быть нескольких видов:

- Диалоговое окно с кнопками;
- Диалоговое окно с кнопками справа;
- Диалоговое окно без кнопок;
- Главное окно;
- Виджет.

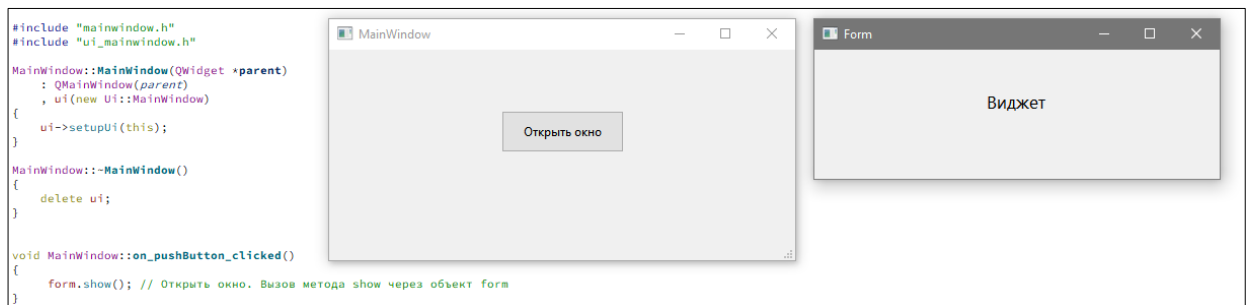
Диалоговые окна подразумевают взаимодействие с пользователем. Они могут быть модалными.



Главное окно может содержать в себе системное меню и строку состояния. Виджет является самым простым окном.



Создадим окно с кнопкой, при нажатии на которую будет открываться виджет, [смотреть](#).



Если кликнуть на кнопку **Открыть окно** еще раз, другое окно **Form** создаваться не будет. Чтобы открыть окно **Form** еще раз, нужно закрыть предыдущее.

Если необходимо каждый раз при клике на кнопку создавать новое окно, то объект класса нужно создавать динамически (указатель на объект).

mainwindow.h

```

#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <form.h> // Подключаем заголовочный файл виджета

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_pushButton_clicked();

private:
    Ui::MainWindow *ui;
    Form* form; // Создание указателя на объект form подключенного класса Form
};
#endif // MAINWINDOW_H
    
```

mainwindow.cpp

```

#include "mainwindow.h"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
}

MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::on_pushButton_clicked()
{
    form = new Form; // Инициализация объекта
    form->show(); // Открыть окно. Вызов метода show через указатель form
}
    
```

Создадим окно с кнопкой, при нажатии на которую будет открываться модальное **Диалоговое окно без кнопок**, [смотреть](#).

Задание к лабораторной работе 2

Для выполнения лабораторной работы необходимо установить и настроить Visual Studio и инструменты Qt. Задание к лабораторной работе 2 состоит из нескольких задач. Задачи выполняются по вариантам (например, Вариант 1 – номера по списку в группе: 1, 9, 17, 26; Вариант 2 – номера по списку в группе: 2, 10, 18, 27 и т.д.).

Вариант 1



Задача 1

За основу необходимо взять класс, созданный для решения задачи в лабораторной работе 1. Каждая строчка кода должна содержать комментарий.

Все поля, описанные в классе необходимо сделать указателями. Добавить в класс конструктор с параметрами, который будет служить для инициализации полей и конструктор без параметров, с любыми значениями. Создать деструктор, который очистит всю занятую память. Описание конструкторов и деструктора должно быть выполнено за пределами класса. Необходимо добавить метод `set`, который будет устанавливать новые значения для полей класса. Метод `set` тоже должен быть описан за пределами класса.

Задача 2

За основу необходимо взять приложение, созданное для решения задачи в лабораторной работе 1. Необходимо дать поясняющие комментарии к коду.

В приложение необходимо добавить название и иконку, соответствующие тематике. Подобрать необходимый размер окна. Добавить кнопку `Подробнее`, при нажатии на которую будет открываться новый виджет с изображением, соответствующим размеру формы. За основу необходимо взять изображение, размещенное сверху. Также необходимо дать название виджета и установить соответствующую иконку.

Вариант 2



Задача 1

За основу необходимо взять класс, созданный для решения задачи в лабораторной работе 1. Каждая строка кода должна содержать комментарий.

Все поля, описанные в классе необходимо сделать указателями. Добавить в класс конструктор с параметрами, который будет служить для инициализации полей и конструктор без параметров, с любыми значениями. Создать деструктор, который очистит всю занятую память. Описание конструкторов и деструктора должно быть выполнено за пределами класса. Необходимо добавить метод `set`, который будет устанавливать новые значения для полей класса. Метод `set` тоже должен быть описан за пределами класса.

Задача 2

За основу необходимо взять приложение, созданное для решения задачи в лабораторной работе 1. Необходимо дать поясняющие комментарии к коду.

В приложение необходимо добавить название и иконку, соответствующие тематике. Подобрать необходимый размер окна. Добавить кнопку `Подробнее`, при нажатии на которую будет открываться новый виджет с изображением, соответствующим размеру формы. За основу необходимо взять изображение, размещенное сверху. Также необходимо дать название виджета и установить соответствующую иконку.

Вариант 3



Задача 1

За основу необходимо взять класс, созданный для решения задачи в лабораторной работе 1. Каждая строчка кода должна содержать комментарий.

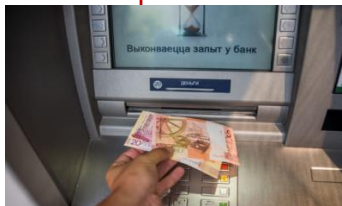
Все поля, описанные в классе необходимо сделать указателями. Добавить в класс конструктор с параметрами, который будет служить для инициализации полей и конструктор без параметров, с любыми значениями. Создать деструктор, который очистит всю занятую память. Описание конструкторов и деструктора должно быть выполнено за пределами класса. Необходимо добавить метод `set`, который будет устанавливать новые значения для полей класса. Метод `set` тоже должен быть описан за пределами класса.

Задача 2

За основу необходимо взять приложение, созданное для решения задачи в лабораторной работе 1. Необходимо дать поясняющие комментарии к коду.

В приложение необходимо добавить название и иконку, соответствующие тематике. Подобрать необходимый размер окна. Добавить кнопку `Подробнее`, при нажатии на которую будет открываться новый виджет с изображением, соответствующим размеру формы. За основу необходимо взять изображение, размещенное сверху. Также необходимо дать название виджета и установить соответствующую иконку.

Вариант 4



Задача 1

За основу необходимо взять класс, созданный для решения задачи в лабораторной работе 1. Каждая строка кода должна содержать комментарий.

Все поля, описанные в классе необходимо сделать указателями. Добавить в класс конструктор с параметрами, который будет служить для инициализации полей и конструктор без параметров, с любыми значениями. Создать деструктор, который очистит всю занятую память. Описание конструкторов и деструктора должно быть выполнено за пределами класса. Необходимо добавить метод `set`, который будет устанавливать новые значения для полей класса. Метод `set` тоже должен быть описан за пределами класса.

Задача 2

За основу необходимо взять приложение, созданное для решения задачи в лабораторной работе 1. Необходимо дать поясняющие комментарии к коду.

В приложение необходимо добавить название и иконку, соответствующие тематике. Подобрать необходимый размер окна. Добавить кнопку `Подробнее`, при нажатии на которую будет открываться новый виджет с изображением, соответствующим размеру формы. За основу необходимо взять изображение, размещенное сверху. Также необходимо дать название виджета и установить соответствующую иконку.

Вариант 5



Задача 1

За основу необходимо взять класс, созданный для решения задачи в лабораторной работе 1. Каждая строчка кода должна содержать комментарий.

Все поля, описанные в классе необходимо сделать указателями. Добавить в класс конструктор с параметрами, который будет служить для инициализации полей и конструктор без параметров, с любыми значениями. Создать деструктор, который очистит всю занятую память. Описание конструкторов и деструктора должно быть выполнено за пределами класса. Необходимо добавить метод `set`, который будет устанавливать новые значения для полей класса. Метод `set` тоже должен быть описан за пределами класса.

Задача 2

За основу необходимо взять приложение, созданное для решения задачи в лабораторной работе 1. Необходимо дать поясняющие комментарии к коду.

В приложение необходимо добавить название и иконку, соответствующие тематике. Подобрать необходимый размер окна. Добавить кнопку `Подробнее`, при нажатии на которую будет открываться новый виджет с изображением, соответствующим размеру формы. За основу необходимо взять изображение, размещенное сверху. Также необходимо дать название виджета и установить соответствующую иконку.

Вариант 6



Задача 1

За основу необходимо взять класс, созданный для решения задачи в лабораторной работе 1. Каждая строка кода должна содержать комментарий.

Все поля, описанные в классе необходимо сделать указателями. Добавить в класс конструктор с параметрами, который будет служить для инициализации полей и конструктор без параметров, с любыми значениями. Создать деструктор, который очистит всю занятую память. Описание конструкторов и деструктора должно быть выполнено за пределами класса. Необходимо добавить метод `set`, который будет устанавливать новые значения для полей класса. Метод `set` тоже должен быть описан за пределами класса.

Задача 2

За основу необходимо взять приложение, созданное для решения задачи в лабораторной работе 1. Необходимо дать поясняющие комментарии к коду.

В приложение необходимо добавить название и иконку, соответствующие тематике. Подобрать необходимый размер окна. Добавить кнопку `Подробнее`, при нажатии на которую будет открываться новый виджет с изображением, соответствующим размеру формы. За основу необходимо взять изображение, размещенное сверху. Также необходимо дать название виджета и установить соответствующую иконку.

Вариант 7



Задача 1

За основу необходимо взять класс, созданный для решения задачи в лабораторной работе 1. Каждая строка кода должна содержать комментарий.

Все поля, описанные в классе необходимо сделать указателями. Добавить в класс конструктор с параметрами, который будет служить для инициализации полей и конструктор без параметров, с любыми значениями. Создать деструктор, который очистит всю занятую память. Описание конструкторов и деструктора должно быть выполнено за пределами класса. Необходимо добавить метод `set`, который будет устанавливать новые значения для полей класса. Метод `set` тоже должен быть описан за пределами класса.

Задача 2

За основу необходимо взять приложение, созданное для решения задачи в лабораторной работе 1. Необходимо дать поясняющие комментарии к коду.

В приложение необходимо добавить название и иконку, соответствующие тематике. Подобрать необходимый размер окна. Добавить кнопку `Подробнее`, при нажатии на которую будет открываться новый виджет с изображением, соответствующим размеру формы. За основу необходимо взять изображение, размещенное сверху. Также необходимо дать название виджета и установить соответствующую иконку.

Вариант 8



Задача 1

За основу необходимо взять класс, созданный для решения задачи в лабораторной работе 1. Каждая строчка кода должна содержать комментарий.

Все поля, описанные в классе необходимо сделать указателями. Добавить в класс конструктор с параметрами, который будет служить для инициализации полей и конструктор без параметров, с любыми значениями. Создать деструктор, который очистит всю занятую память. Описание конструкторов и деструктора должно быть выполнено за пределами класса. Необходимо добавить метод `set`, который будет устанавливать новые значения для полей класса. Метод `set` тоже должен быть описан за пределами класса.

Задача 2

За основу необходимо взять приложение, созданное для решения задачи в лабораторной работе 1. Необходимо дать поясняющие комментарии к коду.

В приложение необходимо добавить название и иконку, соответствующие тематике. Подобрать необходимый размер окна. Добавить кнопку `Подробнее`, при нажатии на которую будет открываться новый виджет с изображением, соответствующим размеру формы. За основу необходимо взять изображение, размещенное сверху. Также необходимо дать название виджета и установить соответствующую иконку.