

МИНОБРНАУКИ РОССИИ

**Федеральное государственное бюджетное образовательное учреждение
высшего образования**

«Тульский государственный университет»

Интернет-институт

ОТЧЕТ ПО ЛАБОРАТОРНЫМ РАБОТАМ

по дисциплине

«Технологии программирования 2»

Семестр 5

Вариант 3

Выполнил: студент гр. ИБ262521-ф

Артемов Александр Евгеньевич

Проверил: канд. техн. наук, доц.

Сафронова Марина Алексеевна

Тула 2024

Лабораторная работа № 1.

Название работы: Введение в объектно-ориентированное программирование.

Цели работы: Получение базовых знаний по реализации ООП в языке C++.

Задачи: Задание на лабораторную работу не выдано. Рассмотрены примеры из теоретического материала.

Пример 1:

Класс пространственного вектора `SpatialVector`. Класс имеет следующие члены — функции:

`void set(double a, double b, double c)` — установка значений координат вектора;

`double abs()` — вычисление абсолютного значения вектора.

Исходный код примера:

```
#include <iostream>
#include <math.h>

using namespace std;

class SpatialVector
{
    double x, y, z;
public:
    void set(double a, double b, double c);
    double abs();
};

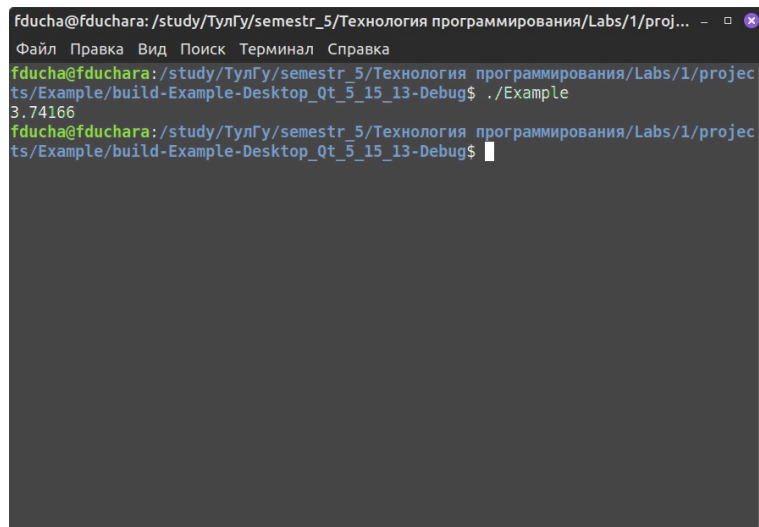
void SpatialVector::set(double a, double b, double c) {
    x = a; y = b; z = c;
}

double SpatialVector::abs() {
    return sqrt(x * x + y * y + z * z);
}

int main() {
    SpatialVector a;
    a.set(1, 2, 3);
    cout << a.abs() << endl;
    return 0;
}
```

При выполнении исходного кода примера создается экземпляр класса `SpatialVector`, задаются значения его координат, вычисляется абсолютное значение вектора и выводится на экран.

Снимок экрана работы программы:



```
fducha@fduchara: /study/ТулГу/semestr_5/Технология программирования/Labs/1/proj...
Файл Правка Вид Поиск Терминал Справка
fducha@fduchara: /study/ТулГу/semestr_5/Технология программирования/Labs/1/projects/Example/build-Example-Desktop_Qt_5_15_13-Debug$ ./Example
3.74166
fducha@fduchara: /study/ТулГу/semestr_5/Технология программирования/Labs/1/projects/Example/build-Example-Desktop_Qt_5_15_13-Debug$
```

Пример 2:

Класс пространственного вектора `SpatialVector`. В класс добавлены методы установки и чтения значения скрытых членов класса (сеттеры и геттеры), которые реализованы внутри класса:

- `void setX(double x)` — метод установки значения координаты `X`;
- `void setY(double y)` — метод установки значения координаты `Y`;
- `void setZ(double z)` — метод установки значения координаты `Z`;
- `double getX() const` — метод чтения значения координаты `X`;
- `double getY() const` — метод чтения значения координаты `Y`;
- `double getZ() const` — метод чтения значения координаты `Z`;

Ключевое слово `const` указывает, что функция является функцией только для чтения, которая не изменяет объект, для которого она вызывается.

В конструктор класса реализован вне класса и выполняет оператор вывода сообщения «Работа конструктора». Деструктор реализован внутри класса и выполняет оператор вывода сообщения «Работа деструктора».

Исходный код примера:

```
#include <iostream>
#include <math.h>

using namespace std;

class SpatialVector
{
    double x, y, z;
public:
    SpatialVector();
    ~SpatialVector() {
        cout << "Работа деструктора" << endl;
    }

    void set(double a, double b, double c);
    double abs();

    double getX() const { return x; }
    void setX(double x) { this->x = x; }
```

```

        double getY() const { return y; }
        void setY(double y) { this->y = y; }
        double getZ() const { return z; }
        void setZ(double z) { this->z = z; }
};

SpatialVector::SpatialVector() {
    cout << "Работа конструктора" << endl;
}

void SpatialVector::set(double a, double b, double c) {
    x = a; y = b; z = c;
}

double SpatialVector::abs() {
    return sqrt(x * x + y * y + z * z);
}

int main() {
    SpatialVector a;
    cout << a.abs() << endl;
    a.setX(3);
    a.setY(4);
    a.setZ(0);
    cout << a.abs() << endl;
    return 0;
}

```

При выполнении исходного кода примера создается экземпляр класса `SpatialVector` (в конструкторе производится вывод сообщения), вычисляется абсолютное значение вектора (по умолчанию объекты класса `double` инициализируются нулем) и выводится на экран (разумеется, 0). Далее, координаты вектора задаются вручную через сеттеры и производится повторное вычисление и вывод абсолютного значения вектора. При выходе из функции `main()` неявно вызывается деструктор класса `SpatialVector` (в деструкторе производится вывод сообщения).

Снимок экрана работы программы:

```

fducha@fduchara: /study/ТулГу/semestr_5/Технология программирования/Labs/1/proj...
Файл Правка Вид Поиск Терминал Справка

fducha@fduchara: /study/ТулГу/semestr_5/Технология программирования/Labs/1/projec
ts/Example/build-Example-Desktop_Qt_5_15_13-Debug$ ./Example
Работа конструктора
0
5
Работа деструктора
fducha@fduchara: /study/ТулГу/semestr_5/Технология программирования/Labs/1/projec
ts/Example/build-Example-Desktop_Qt_5_15_13-Debug$

```

Пример 3:

Класс пространственного вектора `SpatialVector`. В данном примере конструктор по умолчанию заменен на конструктор с параметрами, через которые задаются координаты вектора:

`SpatialVector(double a, double b, double c)` — параметр `a` задает значение координате `X`, параметр `b` — значение координате `Y`, параметр `c` — значение координате `Z`.

Исходный код примера:

```
#include <iostream>
#include <math.h>

using namespace std;

class SpatialVector
{
    double x, y, z;
public:
    SpatialVector(double a, double b, double c);
    ~SpatialVector() {
        cout << "Работа деструктора" << endl;
    }

    //void set(double a, double b, double c);
    double abs();
};

SpatialVector::SpatialVector(double a, double b, double c) {
    x = a; y = b; z = c;
    cout << "Работа конструктора" << endl;
}

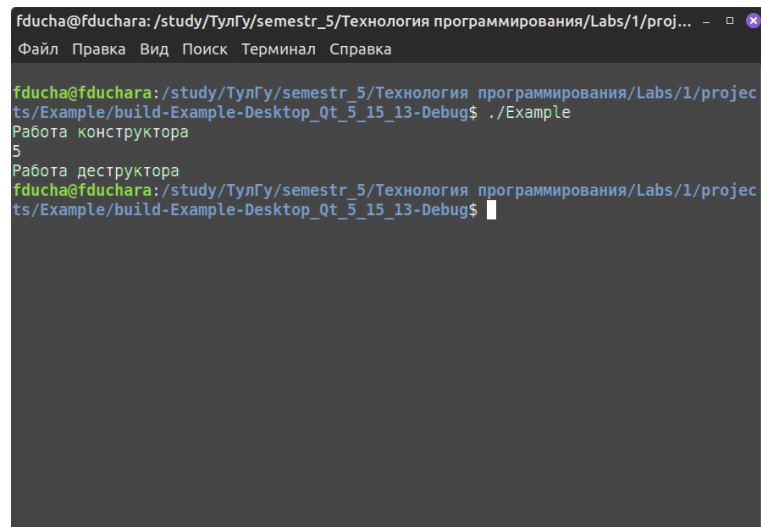
//void SpatialVector::set(double a, double b, double c) {
//    x = a; y = b; z = c;
//}

double SpatialVector::abs() {
    return sqrt(x * x + y * y + z * z);
}

int main() {
    SpatialVector a(3, 4, 0);
    cout << a.abs() << endl;
    return 0;
}
```

При выполнении исходного кода примера создается экземпляр класса `SpatialVector` (в конструкторе производится вывод сообщения) с передачей параметров 3, 4, 0 координатам `X`, `Y` и `Z` соответственно. Вычисляется и выводится абсолютное значение вектора. Объявление и определение член — функции `void set(double a, double b, double c)` закомментировано и не используется. При выходе из функции `main()` неявно вызывается деструктор класса `SpatialVector` (в деструкторе производится вывод сообщения).

Снимок экрана работы программы:



```
fducha@fduchara: /study/ТулГу/semestr_5/Технология программирования/Labs/1/project/Example/build-Example-Desktop_Qt_5_15_13-Debug$ ./Example
Работа конструктора
5
Работа деструктора
fducha@fduchara: /study/ТулГу/semestr_5/Технология программирования/Labs/1/project/Example/build-Example-Desktop_Qt_5_15_13-Debug$
```

Пример 4:

Класс пространственного вектора `SpatialVector`. В данном примере для конструктора с параметрами используются нулевые значения по умолчанию, возвращены сеттеры и геттеры значений координат, реализована член — функция `void info()` для вывода координат вектора, реализована перегрузка оператора сложения для объектов класса:

`SpatialVector operator+ (SpatialVector other)` — член — функция принимает объект `other` класса `SpatialVector`, создает нулевой вектор с и устанавливает ему значения координат через соответствующий сеттер равные значению координат текущего вектора сложенные со значениями координат вектора `other`. Для получения значений координат вызываются геттеры объектов. Возвращает вектор с установленными координатами. Таким образом, перегруженный оператор сложения не изменяет значения объектов и возвращает новый объект.

Исходный код примера:

```
#include <iostream>
#include <math.h>

using namespace std;

class SpatialVector
{
    double x, y, z;
public:
    SpatialVector(double x = 0, double y = 0, double z = 0);
    ~SpatialVector() {
        // cout << "Работа деструктора" << endl;
    }

    double abs();

    double getX() const { return x; }
    void setX(double x) { this->x = x; }
    double getY() const { return y; }
    void setY(double y) { this->y = y; }
```

```

double getZ() const { return z; }
void setZ(double z) { this->z = z; }

void info() {
    cout << "Координаты вектора " << x << ", " << y << ", " << z << endl;
}

SpatialVector operator+ (SpatialVector other) {
    SpatialVector c;
    c.setX(this->getX() + other.getX());
    c.setY(this->getY() + other.getY());
    c.setZ(this->getZ() + other.getZ());
    return c;
}
};

SpatialVector::SpatialVector(double x, double y, double z) {
    this->x = x;
    this->y = y;
    this->z = z;
    // cout << "Работа конструктора" << endl;
}

double SpatialVector::abs() {
    return sqrt(x * x + y * y + z * z);
}

int main() {
    SpatialVector a(1, 2, 3), b(10, 20, 30), c;
    c = a + b;
    c.info();
    return 0;
}

```

При выполнении исходного кода примера создаются два экземпляра класса `SpatialVector`, имеющие имена `a` с координатами 1, 2, 3 и `b` координатами 10, 20, 30, а так же нулевой вектор `c` (используются значения по умолчанию). Вектору `c` присваивается значение суммы векторов `a + b`. Вычисляется и выводится абсолютное значение вектора `c`, посредством член — функции `info()`.

Снимок экрана работы программы:

```

fducha@fduchara: /study/ТулГу/semestr_5/Технология программирования/Labs/1/proj...
Файл Правка Вид Поиск Терминал Справка

fducha@fduchara: /study/ТулГу/semestr_5/Технология программирования/Labs/1/projec
ts/Example/build-Example-Desktop_Qt_5_15_13-Debug$ ./Example
Координаты вектора 11, 22, 33
fducha@fduchara: /study/ТулГу/semestr_5/Технология программирования/Labs/1/projec
ts/Example/build-Example-Desktop_Qt_5_15_13-Debug$

```

Пример 5:

Класс пространственного вектора `SpatialVector`. В данном примере реализована перегрузка операторов ввода и вывода посредством дружественных функций:

`friend ostream& operator<< (ostream &stream, SpatialVector &a)` — объявление дружественной функции перегрузки оператора вывода;

`friend istream& operator>> (istream &stream, SpatialVector &a)` — объявление дружественной функции перегрузки оператора ввода.

Данные функции объявляются как дружественные и определяются вне класса, так как они должны иметь доступ к закрытым полям класса.

Исходный код примера:

```
#include <iostream>
#include <math.h>

using namespace std;

class SpatialVector
{
    double x, y, z;

    friend ostream& operator<< (ostream &stream, SpatialVector &a);
    friend istream& operator>> (istream &stream, SpatialVector &a);
public:
    SpatialVector(double x = 0, double y = 0, double z = 0);

    double abs();
};

SpatialVector::SpatialVector(double x, double y, double z) {
    this->x = x;
    this->y = y;
    this->z = z;
}

double SpatialVector::abs() {
    return sqrt(x * x + y * y + z * z);
}

ostream& operator<< (ostream &stream, SpatialVector &a) {
    stream << "x = " << a.x << ", y = " << a.y << ", z = " << a.z << endl;
    return stream;
}

istream& operator>> (istream &stream, SpatialVector &a) {
    stream >> a.x >> a.y >> a.z;
    return stream;
}

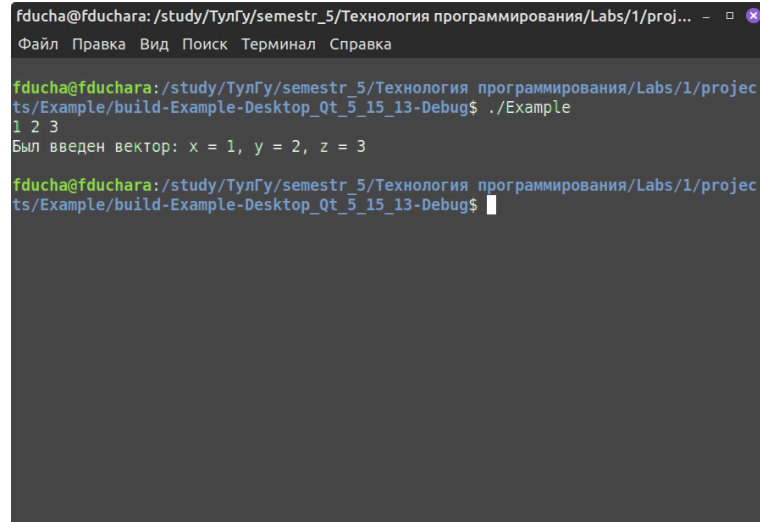
int main() {
    SpatialVector a;
    cin >> a;
    cout << "Был введен вектор: " << a << endl;

    return 0;
}
```

При выполнении исходного кода примера создается нулевой вектор класса `SpatialVector` (используются значения по умолчанию). Значения

координат поступают со стандартного потока ввода с использованием перегруженного оператора ввода. Далее, посредством перегруженного оператора вывода происходит вывод вектора с новыми значениями координат на экран.

Снимок экрана работы программы:

A screenshot of a terminal window with a dark background. The window title is 'fducha@fduchara: /study/ТулГу/semestr_5/Технология программирования/Labs/1/proj...'. The menu bar shows 'Файл', 'Правка', 'Вид', 'Поиск', 'Терминал', and 'Справка'. The terminal shows the command 'fducha@fduchara: /study/ТулГу/semestr_5/Технология программирования/Labs/1/projects/Example/build-Example-Desktop_Qt_5_15_13-Debug\$./Example' and its output: '1 2 3' followed by 'Был введен вектор: x = 1, y = 2, z = 3'. The prompt 'fducha@fduchara: /study/ТулГу/semestr_5/Технология программирования/Labs/1/projects/Example/build-Example-Desktop_Qt_5_15_13-Debug\$' is visible at the bottom.

Пример 6:

Класс точки на плоскости Point. В данном примере реализован класс точки на плоскости, имеющий координаты X и Y, а так же дружественная функция поиска точки в массиве:

friend void findPoint(Point *first, Point *last, Point arg) — first — указатель на начало массива, last — указатель на последний элемент массива, arg — искомая точка. В зависимости от результатов поиска функция выводит сообщение найдена ли искомая точка или нет и координаты искомой точки.

Исходный код примера:

```
#include <iostream>

using namespace std;

class Point
{
private:
    int x, y;
    friend void findPoint(Point *first, Point *last, Point arg);
public:
    Point(int x, int y) {
        this->x = x;
        this->y = y;
    }
};

void findPoint(Point *first, Point *last, Point arg) {
    for (auto *p = first; p <= last; p++) {
        if ((p->x == arg.x) && (p->y == arg.y)) {
            cout << "Точка с координатами " << p->x << ", " << p->y <<
                " найдена" << endl;
            return;
        }
    }
}
```

```

    }
}
cout << "Точка с координатами " << arg.x << ", " << arg.y <<
      " не найдена" << endl;
}

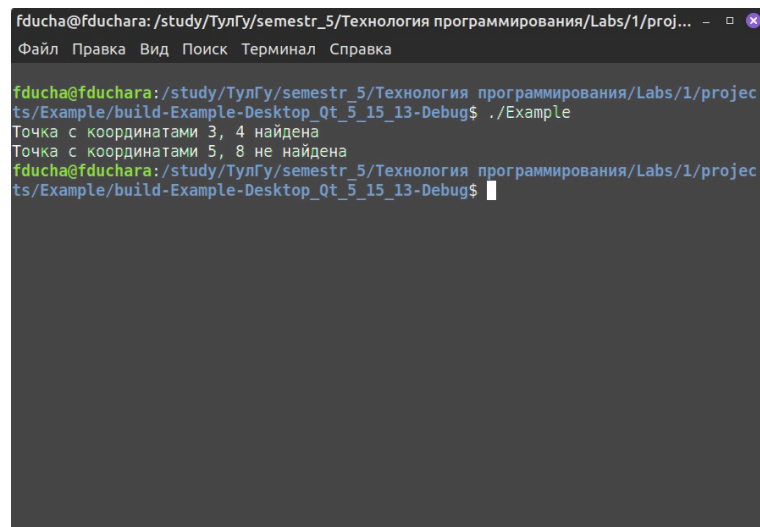
int main()
{
    Point *points[10];
    for (int i = 0; i < 10; i++) {
        points[i] = new Point(i, i+1);
    }
    findPoint(points[0], points[9], Point(3, 4));
    findPoint(points[0], points[9], Point(5, 8));

    return 0;
}

```

При выполнении исходного кода примера создается массив из 10 указателей на объекты класса `Point`, в цикле создаются объекты точек с координатами (0, 1), (1, 2), (3, 4) и так далее, массив заполняется указателями на точки. Функция поиска точки запускается дважды: для поиска точки, имеющейся в массиве (3, 4) и отсутствующей в массиве (5, 8). Следовательно, программа находит первую точку и не находит вторую.

Снимок экрана работы программы:



```

fducha@fduchara: /study/ТулГу/semestr_5/Технология программирования/Labs/1/proj...
Файл Правка Вид Поиск Терминал Справка

fducha@fduchara: /study/ТулГу/semestr_5/Технология программирования/Labs/1/projec
ts/Example/build-Example-Desktop_Qt_5_15_13-Debug$ ./Example
Точка с координатами 3, 4 найдена
Точка с координатами 5, 8 не найдена
fducha@fduchara: /study/ТулГу/semestr_5/Технология программирования/Labs/1/projec
ts/Example/build-Example-Desktop_Qt_5_15_13-Debug$

```

Пример 7:

Класс точки на плоскости `Point`. В данном примере к реализации класса добавлена статическая переменная счетчика количества существующих объектов класса `static int count`. Данная статическая переменная увеличивается на единицу в конструкторе класса при создании нового объекта и уменьшается на единицу в деструкторе при уничтожении.

В исходном коде примера присутствует неточность: при создании и уничтожении объектов выводится сообщение «Создается (Удаляется) точка номер ...». В действительности выводится не номер объекта, а их количество, к тому же, нельзя определить в каком порядке для объектов будет

вызываться деструктор, поэтому текст сообщения изменен на «Количество объектов ...», а оператор декремента в деструкторе класса изменен на префиксный для вывода количества объектов, оставшихся после работы деструктора. Функция `findPoint` не используется.

Исходный код примера:

```
#include <iostream>

using namespace std;

class Point
{
private:
    int x, y;
    static int count;
    friend void findPoint(Point *first, Point *last, Point arg);
public:
    Point(int x = 0, int y = 0) {
        this->x = x;
        this->y = y;

        cout << "Количество объектов " << ++count << endl;
    }
    ~Point() {
        cout << "Количество объектов " << --count << endl;
    }
};

int Point::count;

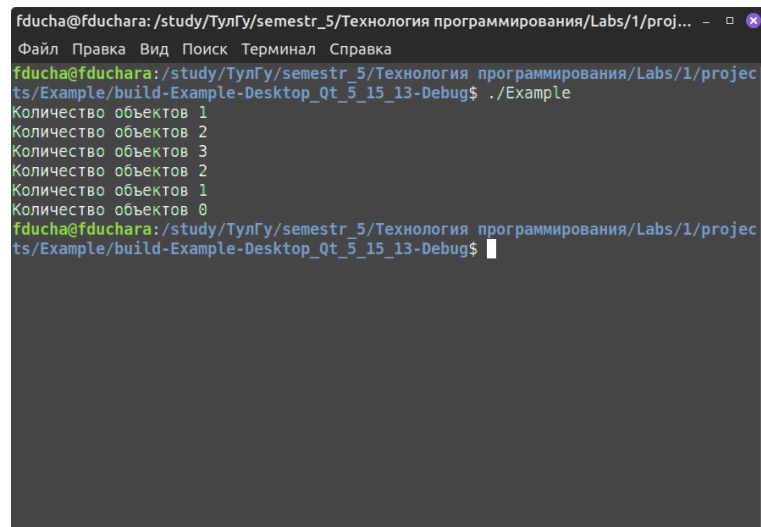
void findPoint(Point *first, Point *last, Point arg) {
    for (auto *p = first; p <= last; p++) {
        if ((p->x == arg.x) && (p->y == arg.y)) {
            cout << "Точка с координатами " << p->x << ", " << p->y << "
найдена" << endl;
            return;
        }
    }
    cout << "Точка с координатами " << arg.x << ", " << arg.y << " не найдена"
<< endl;
}

int main()
{
    Point a, b, c;

    return 0;
}
```

При выполнении исходного кода примера создаются 3 объекта класса `Point`, которые уничтожаются неявным вызовом деструктора при выходе из функции `main()`. При выполнении кода конструктора и деструктора происходит вывод сообщения о количестве существующих объектов класса `Point`.

Снимок экрана работы программы:



```
fducha@fduchara: /study/ТулГу/semestr_5/Технология программирования/Labs/1/proj...
Файл Правка Вид Поиск Терминал Справка
fducha@fduchara: /study/ТулГу/semestr_5/Технология программирования/Labs/1/projec
ts/Example/build-Example-Desktop_Qt_5_15_13-Debug$ ./Example
Количество объектов 1
Количество объектов 2
Количество объектов 3
Количество объектов 2
Количество объектов 1
Количество объектов 0
fducha@fduchara: /study/ТулГу/semestr_5/Технология программирования/Labs/1/projec
ts/Example/build-Example-Desktop_Qt_5_15_13-Debug$
```

Пример 8:

Класс целого числа Integer. В данном примере реализована обертка над целочисленным типом int с перегруженными постфиксным и префиксным операторами инкремента. В перегруженных функциях добавлен вывод сообщения какой оператор вызван, постфиксный или префиксный.

Префиксный оператор увеличивает на единицу закрытую переменную value типа int, выводит сообщение и возвращает ссылку на себя.

Постфиксный оператор изменен, так как код примере действует аналогичного выполнению префиксного оператора. В измененном коде создается временный объект копированием текущего объекта, выполняется префиксный инкремент текущего объекта, выводится сообщение, возвращается временный объект. В итоге оператор возвращает свою копию до инкремента.

Для демонстрации работы класса добавлен геттер getValue() закрытой переменной value.

Исходный код примера:

```
#include <iostream>

using namespace std;

class Integer
{
    int value;
public:
    Integer() { value = 0; }

    Integer& operator++();
    Integer operator++(int);

    int getValue() const { return value; }
};

Integer &Integer::operator++() {
    value += 1;
    cout << "Использован префиксный оператор" << endl;
```

```

        return *this;
    }

    // было
    //Integer &Integer::operator++(int) {
    //    value += 1;
    //    cout << "Использован постфиксный оператор" << endl;
    //    return *this;
    //}

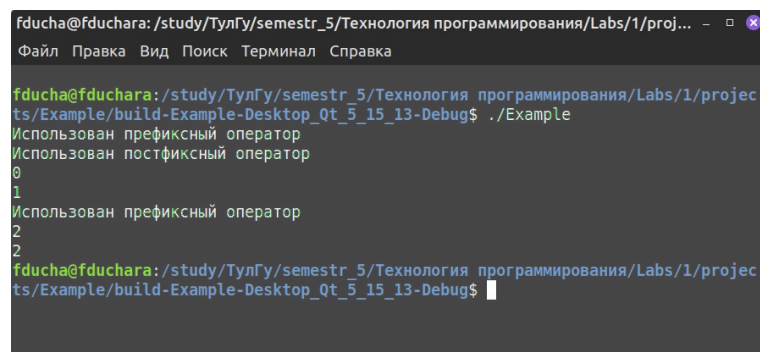
    // стало
    Integer Integer::operator++(int) {
        Integer t = *this;
        ++*this;
        cout << "Использован постфиксный оператор" << endl;
        return t;
    }

    int main()
    {
        Integer i, i2;
        int a;
        i2 = i++;
        a = i2.getValue();
        cout << a << endl;
        a = i.getValue();
        cout << a << endl;
        i2 = ++i;
        a = i2.getValue();
        cout << a << endl;
        a = i.getValue();
        cout << a << endl;
        return 0;
    }

```

При выполнении исходного кода примера создаются 2 объекта класса Integer, i и i2, а так же целочисленная переменная a. В переменную i2 записывается результат выполнения постфиксного инкремента i (при создании i значение равно 0). Значение i2 равно 0, а i — 1 (выполнен инкремент). Далее в переменную i2 записывается результат выполнения префиксного инкремента i. Значение i и i2 равно 2 (инкремент выполнен до копирования).

Снимок экрана работы программы:



```

fducha@fduchara: /study/ТулГу/semestr_5/Технология программирования/Labs/1/proj...
Файл Правка Вид Поиск Терминал Справка

fducha@fduchara: /study/ТулГу/semestr_5/Технология программирования/Labs/1/projec
ts/Example/build-Example-Desktop_Qt_5_15_13-Debug$ ./Example
Использован префиксный оператор
Использован постфиксный оператор
0
1
Использован префиксный оператор
2
2
fducha@fduchara: /study/ТулГу/semestr_5/Технология программирования/Labs/1/projec
ts/Example/build-Example-Desktop_Qt_5_15_13-Debug$

```

Пример 9:

Класс будильника Alarm. В данном примере реализуется класс, содержащий статическую функцию, возвращающую текущее время.

Для выполнения операций на данных о текущем системном времени подключен заголовочный файл time.h.

Исходный код примера:

```
#include <iostream>
#include <time.h>

using namespace std;

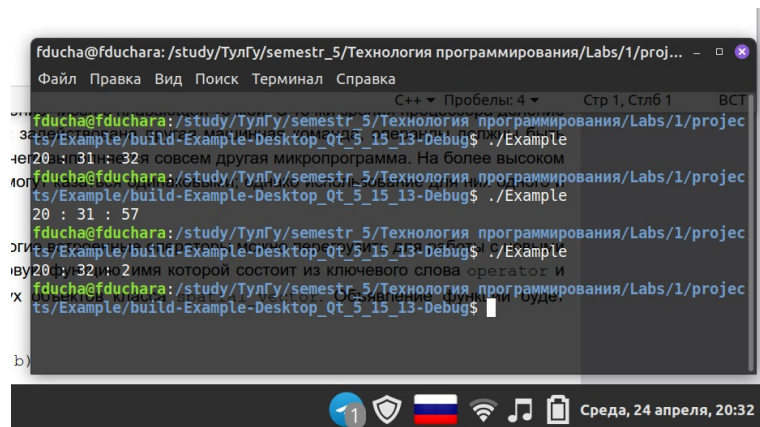
class Alarm
{
//    time_t alarm_t;
public:
    static void currentTime();
};

void Alarm::currentTime() {
    time_t t = time(NULL);
    struct tm tm = *localtime(&t);
    cout << tm.tm_hour << " : " << tm.tm_min << " : " << tm.tm_sec << endl;
}

int main()
{
    Alarm::currentTime();
    return 0;
}
```

При выполнении исходного кода примера вызывается статическая функция currentTime() класса Alarm. Функция выводит сообщение о текущем системном времени в формате «час : минута : секунда».

Снимок экрана работы программы:



Пример 10:

Класс прямоугольной матрицы `Matrix`. В данном примере реализуется класс двумерной матрицы вещественных чисел, заданной размерности.

Конструктор класса принимает два неотрицательных целых аргумента, содержащие количество столбцов и строк матрицы, и создает одномерный массив вещественных чисел нужной размерности для хранения значений элементов матрицы. Реализован конструктор копирования для недопущения ситуаций, приведенных в примере теоретического материала, так как при использовании конструктора копирования по умолчанию несколько матриц могут обращаться к одному массиву значений (копируется указатель на массив, а не значения массива).

Класс имеет методы установки и чтения значений матрицы, элемент определяется по номерам столбца и строки. В сеттере реализована проверка выхода из диапазона размерности матрицы.

Деструктор класса выполняет удаление массива значений.

Исходный код примера:

```
#include <iostream>

using namespace std;

class Matrix
{
    double *m;
    size_t width, height;
public:
    Matrix(size_t w, size_t h);
    Matrix(const Matrix &other);
    ~Matrix();

    double getValue(size_t i, size_t j);
    void setValue(size_t i, size_t j, double val);
};

Matrix::Matrix(size_t w, size_t h) {
    m = new double[w * h];
    width = w;
    height = h;
}

Matrix::Matrix(const Matrix &other) {
    m = new double[other.width * other.height];
    width = other.width;
    height = other.height;
}

Matrix::~Matrix() {
    delete [] m;
}

double Matrix::getValue(size_t i, size_t j) {
    return m[i * width + j];
}

void Matrix::setValue(size_t i, size_t j, double val) {
    if ((i < width) && (j < height)) {
        m[i * width + j] = val;
    }
}
```

```

}

int main()
{
    Matrix a(2, 2);
    a.setValue(0, 0, 100);
    Matrix b = a;
    b.setValue(0, 0, 200);
    cout << "a[0, 0] = " << a.getValue(0, 0) << "; b[0, 0] = " <<
        b.getValue(0, 0) << endl;
    return 0;
}

```

При выполнении исходного кода примера создается матрица *a* размерностью 2 x 2, элементу с индексами 0, 0 устанавливается значение 100. Создается матрица *b* копированием матрицы *a*, элементу с индексами 0, 0 устанавливается значение 200.

При использовании конструктора копирования по умолчанию элементы с индексами 0, 0 обеих матриц равны 200, так как матрицы обращаются к одному массиву значений. При неявном вызове деструктора происходит двойная попытка освобождения памяти (обе матрицы удаляют один и тот же массив значений).

При реализации собственного конструктора копирования матрицы имеют различные массивы значений. Ошибки двойного освобождения памяти не происходит.

Снимок экрана работы программы с конструктором копирования по умолчанию:

```

fducha@fduchara: /study/ТулГу/semestr_5/Технология программирования/Labs/1/proj...
Файл Правка Вид Поиск Терминал Справка

fducha@fduchara: /study/ТулГу/semestr_5/Технология программирования/Labs/1/projec
ts/Example/build-Example-Desktop_Qt_5_15_13-Debug$ ./Example
a[0, 0] = 200; b[0, 0] = 200
free(): double free detected in tcache 2
Аварийный останов (образ памяти сброшен на диск)
fducha@fduchara: /study/ТулГу/semestr_5/Технология программирования/Labs/1/projec
ts/Example/build-Example-Desktop_Qt_5_15_13-Debug$

```

Снимок экрана работы программы с собственной реализацией конструктора копирования:

```

fducha@fduchara: /study/ТулГу/semestr_5/Технология программирования/Labs/1/proj...
Файл Правка Вид Поиск Терминал Справка

fducha@fduchara: /study/ТулГу/semestr_5/Технология программирования/Labs/1/projec
ts/Example/build-Example-Desktop_Qt_5_15_13-Debug$ ./Example
a[0, 0] = 100; b[0, 0] = 200
fducha@fduchara: /study/ТулГу/semestr_5/Технология программирования/Labs/1/projec
ts/Example/build-Example-Desktop_Qt_5_15_13-Debug$

```


Выводы.

Примеры данной лабораторной работы выполнены при помощи интегрированной среды разработки (IDE) Qt 5.15.3, редактора кода Qt Creator 6.0.2, компилятора g++ 11.4.0 (стандарт c++17), системы сборки qmake на операционной системе Linux Mint 21.3 Cinnamon.

C++ является современным и мощным инструментом разработки программного обеспечения, реализующим классическую структурную, объектно-ориентированную и функциональную парадигму. Существует возможность расширения встроенных инструментов для функционального стиля путем подключения сторонних библиотек «реактивного» программирования. Компилятор языка существует практически для всех операционных систем. Исполняемые файлы программ на C++ имеют небольшой размер и наивысшую скорость выполнения, не требуют интерпретатора.

В примерах рассмотрена реализация принципов объектно-ориентированного программирования (ООП) при помощи языка C++, например, создание, копирование и удаление объектов, получение доступа к закрытым членам класса через геттеры и сеттеры или дружественные функции, статические объекты и функции, перегрузка операторов.

Схемы алгоритмов примеров не представлены ввиду тривиальности основного кода примеров (содержание функции main) и прямой последовательности выполнения операций.

Исправлены некоторые неточности в исходном коде примеров.

Лабораторная работа № 2.

Название работы: Наследование классов.

Цели работы: Изучение принципов наследования классов в языке C++.

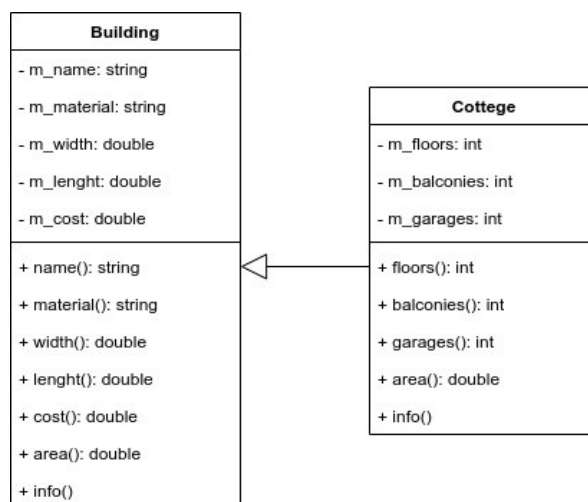
Задачи:

1. Определить иерархию наследования из двух классов в соответствии с номером задания. Каждый класс снабдить свойствами и методами в соответствии с предметной областью, указанной в варианте задания. В базовом классе предусмотреть метод `info()`, выводящий на экран информацию об объекте. Предусмотреть конструкторы, инициализирующие свойства объектов переданными данными либо значениями по умолчанию. Написать демонстрационную программу, создающую 4-5 объектов и выводящую на экран информацию о них. Варианты классов: «Здание», «Коттедж».

2. Реализовать класс, содержащий коллекцию объектов, методы для включения и удаления элементов, вывода содержимого коллекции на экран, а также перегруженный в соответствии с заданием оператор. Написать программу, заполняющую коллекцию несколькими элементами и демонстрирующую пользователю работу перегруженного оператора для элементов коллекции: «*» (пересечение коллекций), класс «множество целых чисел».

Задача № 1.

Схема алгоритма задания.



Класс «Здание» (Building) имеет такие скрытые поля такие, как:

`m_name` — имя здания, тип `string`;

`m_material` — материал здания, тип `string`;

`m_width` — ширина здания, тип `double`;

`m_lenght` — длина здания, тип `double`;

m_cost — стоимость здания, тип double.

Так же класс Building имеет открытые геттеры к закрытым полям класса, имеющие соответствующие имена и методы как:

area() — возвращает площадь здания;

info() — выводит сообщение информацию о здании на основании данных, содержащихся в скрытых полях класса.

Значения скрытых полей устанавливаются при создании объекта путем передачи данных в конструктор класса.

Класс «Коттедж» (Cottege) расширяет класс Building и имеет собственные скрытые поля такие, как:

m_floors — количество этажей, тип int;

m_balconies — количество балконов, тип int;

m_garages — количество гаражей, тип int.

Для данных скрытых полей реализованы соответствующие геттеры.

Класс Cottege переопределяет методы area() и info() базового класса Building. При переопределении используются одноименные методы базового класса.

Конструктор класса Cottege использует конструктор базового класса для инициализации скрытых полей. Для инициализации собственных скрытых полей в конструкторе класса Cottege для них указаны значения по умолчанию.

Текст программы.

```
#include <iostream>
#include <iomanip>

using namespace std;

class Building
{
public:
    Building(const string &name, const string &material, double width,
             double lenght, double cost) :
        m_name(name), m_material(material), m_width(width), m_lenght(lenght),
        m_cost(cost) {}

    string name() const { return m_name; }
    string material() const { return m_material; }
    double width() const { return m_width; }
    double lenght() const { return m_lenght; }
    double cost() const { return m_cost; }

    double area() const { return m_width * m_lenght; }

    void info() {
        cout << "_____ " << endl;
        cout << name() << endl;
        cout << "Материал: " << material() << endl;
        cout << "Ширина: " << width() << " м." << endl;
        cout << "Длина: " << lenght() << " м." << endl;
        cout << "Цена: " << fixed << setprecision(0) << cost() << " руб."
            << endl;
        cout << "Площадь: " << area() << " кв. м." << endl;
    }
}
```

```

private:
    string m_name;
    string m_material;
    double m_width;
    double m_lenght;
    double m_cost;
};

class Cottege : public Building
{
public:
    Cottege(const string &name, const string &material, double width,
            double lenght, double cost,
            int floors = 1, int balconies = 0, int garages = 0) :
        Building(name, material, width, lenght, cost),
        m_floors(floors), m_balconies(balconies), m_garages(garages) {}

    double area() const {
        if (floors() > 1) return Building::area() * m_floors;
        else return Building::area();
    }

    void info() {
        cout << "===== " << endl;
        cout << "Коттедж ";
        Building::info();

        if (floors() > 1) {
            cout << floors() << " этажа" << endl;
            cout << "Площадь " << floors() << " этажей: " << area()
                << " кв. м.";
        }
        else cout << "Одноэтажный";
        cout << endl;

        if (balconies() >= 1) cout << balconies() << " балкон(а)";
        else cout << "Балконы отсутствуют.";
        cout << endl;

        if (garages() >= 1) cout << garages() << " гараж(а)";
        else cout << "Гаражи отсутствуют.";
        cout << endl;
    }

    int floors() const { return m_floors; }

    int balconies() const { return m_balconies; }

    int garages() const { return m_garages; }

private:
    int m_floors;
    int m_balconies;
    int m_garages;
};

int main()
{
    Building b1("Склад", "Пеноблок", 30, 50, 15000000);
    b1.info();
    Building b2("Жилой дом", "Бетон", 20, 40, 33600000);
    b2.info();
    Cottege c1("Универсальный", "Кирпич", 14, 10, 3000000, 2, 4);
    c1.info();
    Cottege c2("Сельский", "Брус", 5, 7, 1500000, 1, 0, 1);
}

```

```

        c2.info();
        return 0;
    }

```

При выполнении программы создаются 2 объекта класса Building и 2 объекта класса Cottage. Объектам класса Building при создании передаются: имя здания, тип материала, ширина, длина и стоимость. При выводе информации о здании при помощи метода info() выводятся указанные значения, а так же вычисленная площадь здания. При создании объектов класса Cottage помимо имени здания, типа материала, ширины, длины и стоимости опционально указываются количество этажей, балконов и гаражей. По умолчанию количество этажей равно 1, балконов и гаражей — 0.

При количестве этажей равному 1 при вызове метода info() выводится значение «Одноэтажный». При отсутствии гаражей и балконов выводятся значения «Гаражи отсутствуют» и «Балконы отсутствуют». В иных случаях выводится информация о количестве этажей, балконов и гаражей.

В переопределенном методе area() класса Cottage расчет площади производится с учетом количества этажей.

Снимок экрана результатов выполнения задания.

```

fducha@fduchara: /study/ТулГу/semestr_5/Технология программирования/Labs/2/proj...
Файл Правка Вид Поиск Терминал Справка

fducha@fduchara: /study/ТулГу/semestr_5/Технология программирования/Labs/2/projec
t/build-Building-Desktop_Qt_5_15_13-Debug$ ./Building

Склад
Материал: Пеноблок
Ширина: 30 м.
Длина: 50 м.
Цена: 15000000 руб.
Площадь: 1500 кв. м.

Жилой дом
Материал: Бетон
Ширина: 20 м.
Длина: 40 м.
Цена: 33600000 руб.
Площадь: 800 кв. м.

=====
Коттедж
Универсальный
Материал: Кирпич
Ширина: 14 м.
Длина: 10 м.
Цена: 3000000 руб.
Площадь: 140 кв. м.
2 этажа
Площадь 2 этажей: 280 кв. м.
4 балкон(а)
Гаражи отсутствуют.

=====
Коттедж
Сельский
Материал: Брус
Ширина: 5 м.
Длина: 7 м.
Цена: 1500000 руб.
Площадь: 35 кв. м.
Одноэтажный
Балконы отсутствуют.
1 гараж(а)

fducha@fduchara: /study/ТулГу/semestr_5/Технология программирования/Labs/2/projec
t/build-Building-Desktop_Qt_5_15_13-Debug$

```

Задача № 2.

Схема алгоритма задания.

IntCollection
- m_set: set<int>
+ add(int)
+ remove(int)
+ print()
+ operator*(const IntCollection&): IntCollection

Класс множества целых чисел (IntCollection) имеет скрытые поля:
m_set — множество целых, тип set<int> (необходимо подключение заголовочного файла <set>).

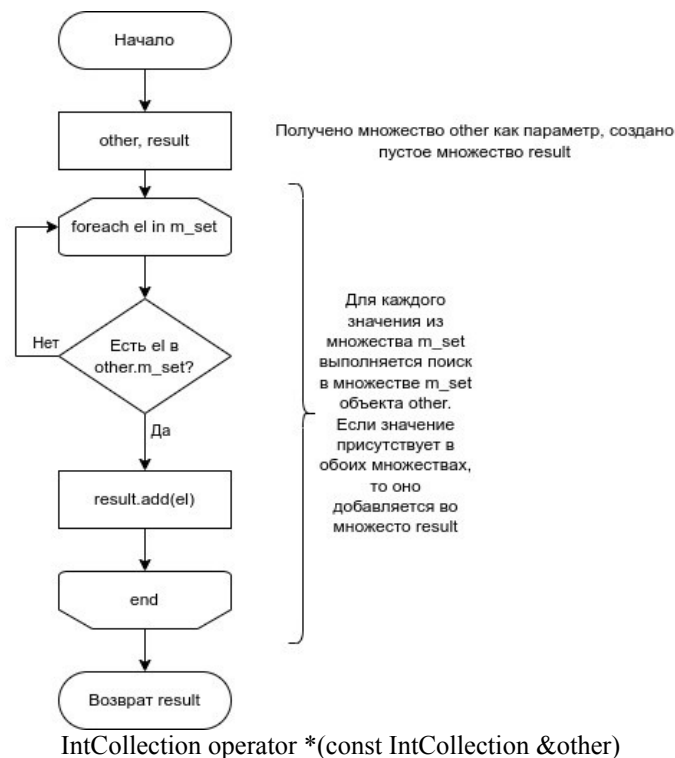
Методы класса IntCollection:

add(int) — добавляет значение в коллекцию;

remove(int) — удаляет значение из коллекции;

operator*(const IntCollection&) — перегрузка оператора умножения, используется для вычисления пересечения множеств, возвращает новый объект класса IntCollection;

print() — выводит элементы коллекции в виде строки.



Алгоритм работы вычисления пересечения коллекций: перегруженной функции умножения передается константная ссылка на правый операнд other. Создается пустая коллекция result, в которую будет записываться результат пересечения. Для каждого элемента скрытого поля m_set текущего объекта (левого операнда) в цикле выполняется поиск равного элемента в скрытом поле m_set правого операнда. Если элемент есть в обоих

коллекциях, то он добавляется в коллекцию result. После выполнения цикла функция возвращает коллекцию result.



Алгоритм работы программы: создается коллекция A и заполняется целыми числами от 1 до 7. Далее для демонстрации удаляется элемент, имеющий значение 1, и коллекция A выводится на экран. Аналогично создается коллекция B и заполняется целыми числами от 6 до 15, удаляется элемент, имеющий значение 13, и коллекция B выводится на экран.

Создается коллекция I путем копирования результата пересечения коллекций A и B. Коллекция I выводится на экран.

Текст программы.

```
#include <iostream>
#include <set>

using namespace std;

class IntCollection
{
public:
    IntCollection() {}
    void add(int element) { m_set.insert(element); }
    void remove(int element) { m_set.erase(element); }
    void print() {
        for (int el : m_set) {
            cout << el << " ";
        }
        cout << endl;
    };
    IntCollection operator *(const IntCollection &other) {
        IntCollection result;
        for (int el : this->m_set) {
            if (other.m_set.count(el) > 0) result.add(el);
        }
        return result;
    }
private:
    set<int> m_set;
};

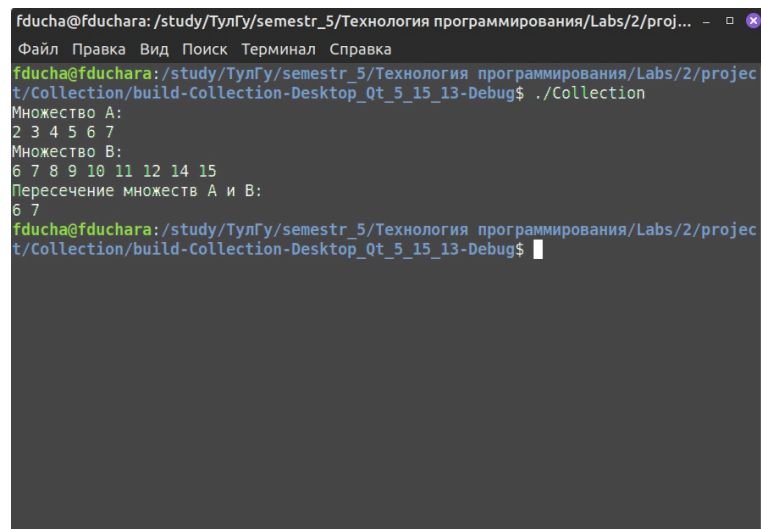
int main()
{
    IntCollection A;
    for (int i = 1; i <= 7; i++) {
        A.add(i);
    }
    A.remove(1);
    cout << "Множество A:" << endl;
    A.print();

    IntCollection B;
    for (int i = 6; i <= 15; i++) {
        B.add(i);
    }
    B.remove(13);
    cout << "Множество B:" << endl;
    B.print();

    cout << "Пересечение множеств A и B:" << endl;
    IntCollection I = A * B;
    I.print();

    return 0;
}
```


Снимок экрана результатов выполнения задания.



```
fducha@fduchara: /study/ТулГу/semestr_5/Технология программирования/Labs/2/proj... - □ ×
Файл Правка Вид Поиск Терминал Справка
fducha@fduchara: /study/ТулГу/semestr_5/Технология программирования/Labs/2/projec
t/Collection/build-Collection-Desktop_Qt_5_15_13-Debug$ ./Collection
Множество A:
2 3 4 5 6 7
Множество B:
6 7 8 9 10 11 12 14 15
Пересечение множеств A и B:
6 7
fducha@fduchara: /study/ТулГу/semestr_5/Технология программирования/Labs/2/projec
t/Collection/build-Collection-Desktop_Qt_5_15_13-Debug$ █
```

Выводы по работе.

Язык программирования C++ в полной мере позволяет реализовать наследование пользовательских классов, расширяя функционал базовых классов новыми полями и методами, или изменять поведение методов базового класса путем перегрузки или использованием виртуальных методов.

Так же C++ позволяет переопределять поведение существующих операторов путем их перегрузки.

Лабораторная работа № 3.

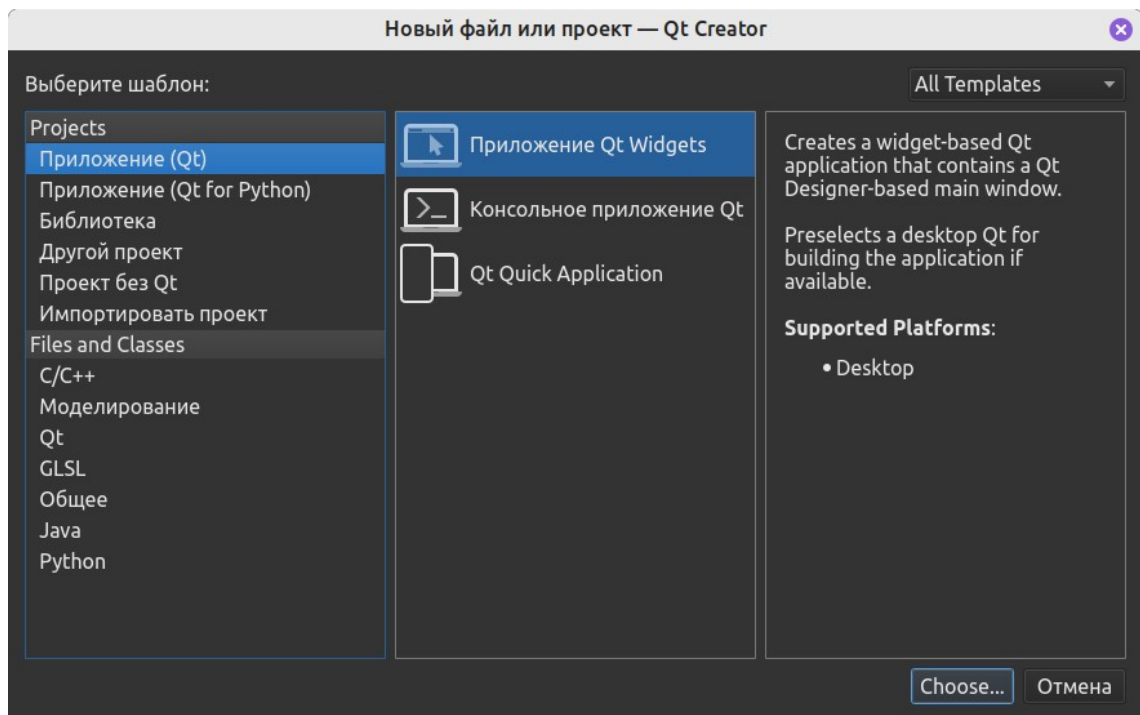
Название работы: Знакомство с Qt.

Цели работы: Изучение настроек среды Qt Creator.

Задачи:

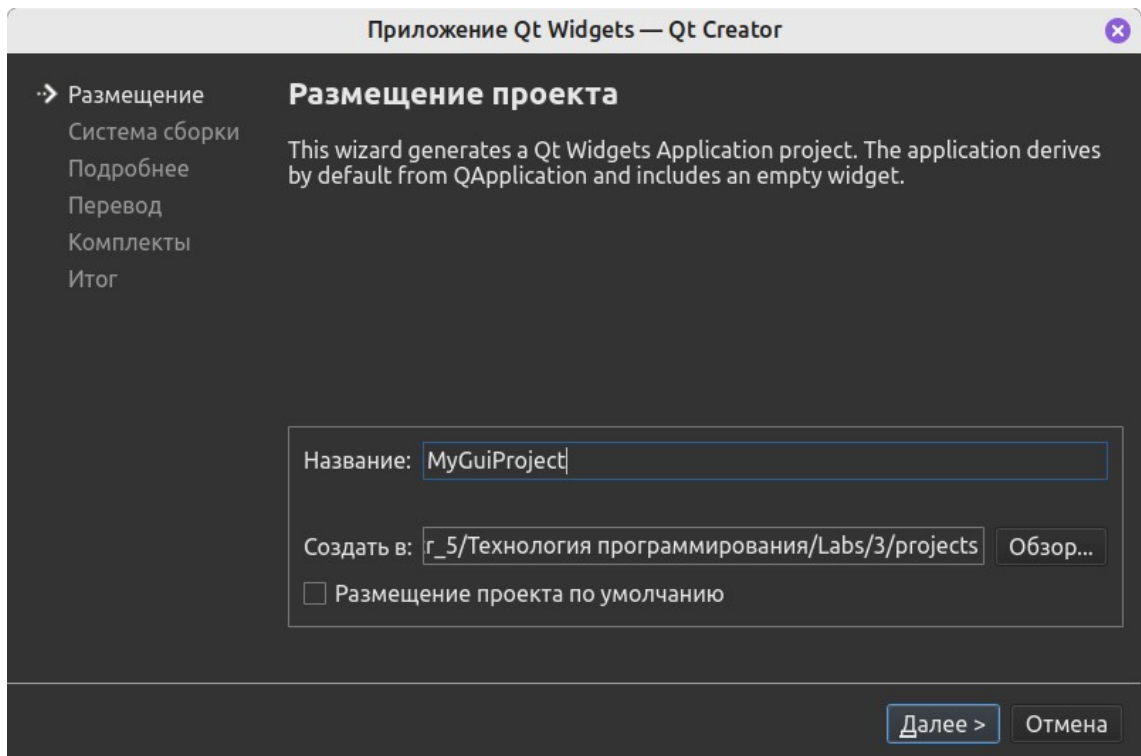
1. Повторите описанные шаги для создания собственного проекта. Назовите проект MyGuiProject. Класс главного окна программы назовите MyMainWindow. Просмотрите структуру проекта. Скомпилируйте и запустите проект на выполнение.
2. Попробуйте использовать любую из горячих клавиш, описанных в таблице «Некоторые важные горячие клавиши».
3. Используйте документацию Qt и найдите описание для классов QMainWindow и QApplication.

1. В запущенной среде Qt Creator открываем меню Файл → Создать файл или проект...

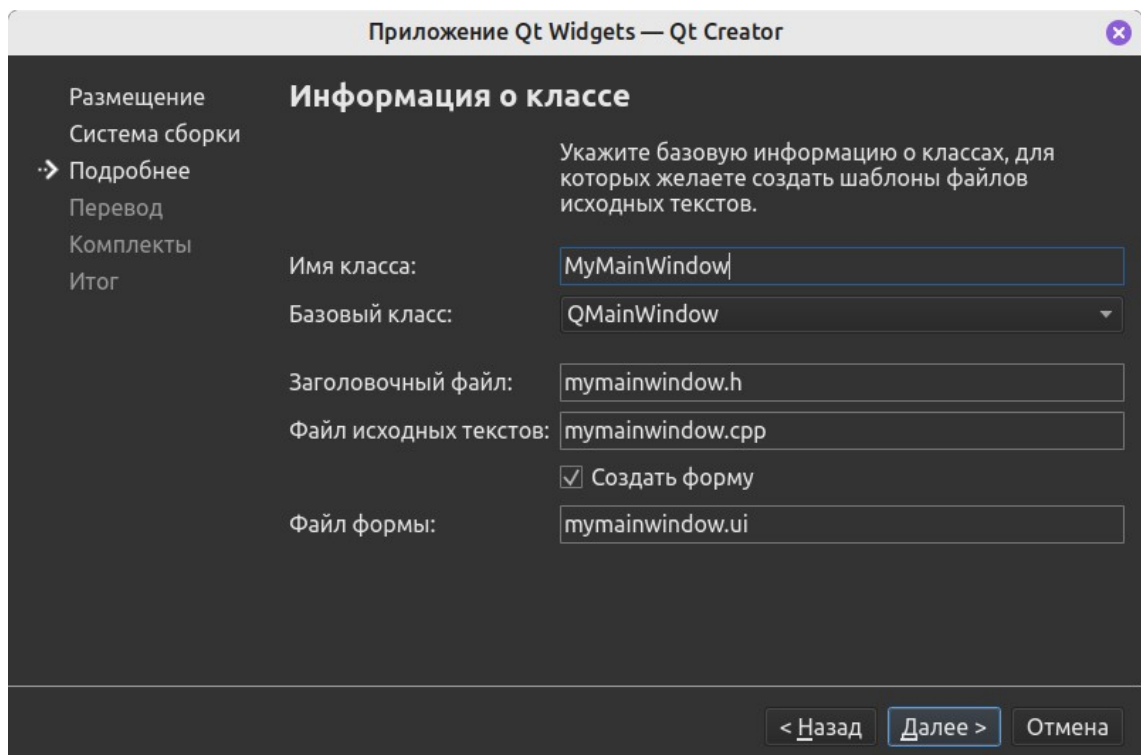


В появившемся окне выбираем шаблон Приложение (Qt) в разделе Projects, далее выбираем Приложение Qt Widgets, нажимаем кнопку Choose...

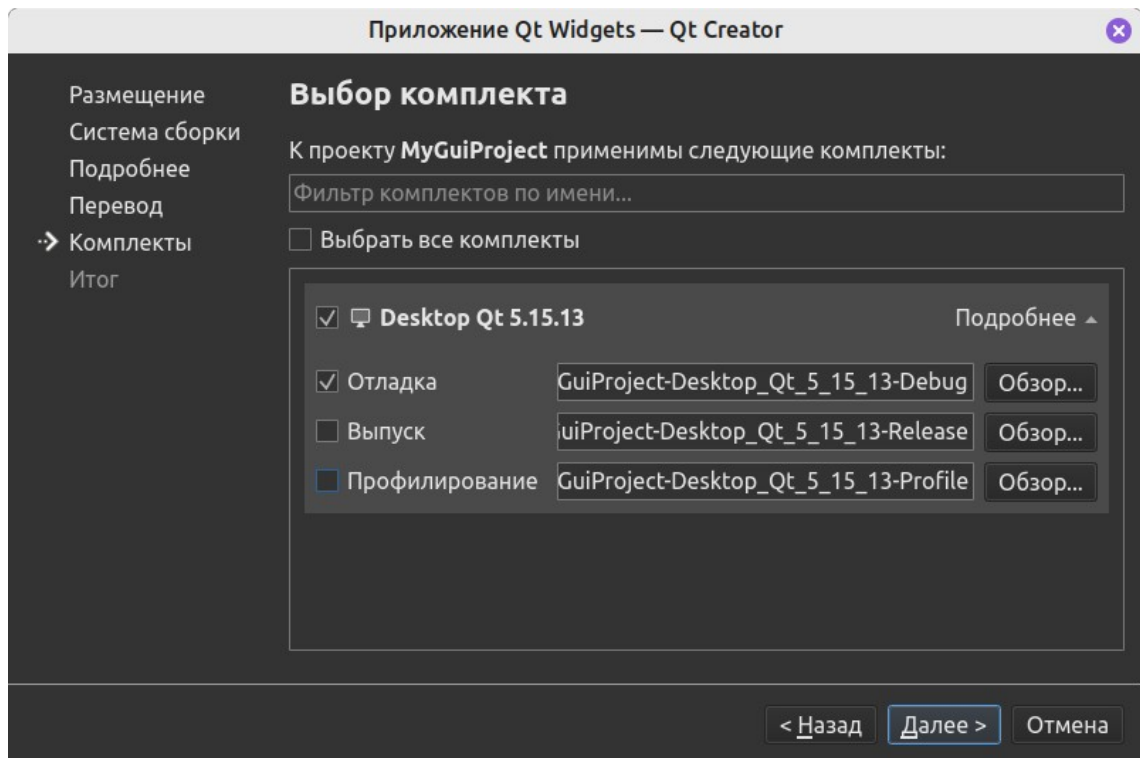
Указываем каталог для размещения проекта и вводим название проекта MyGuiProject. Нажимаем кнопку Далее >.



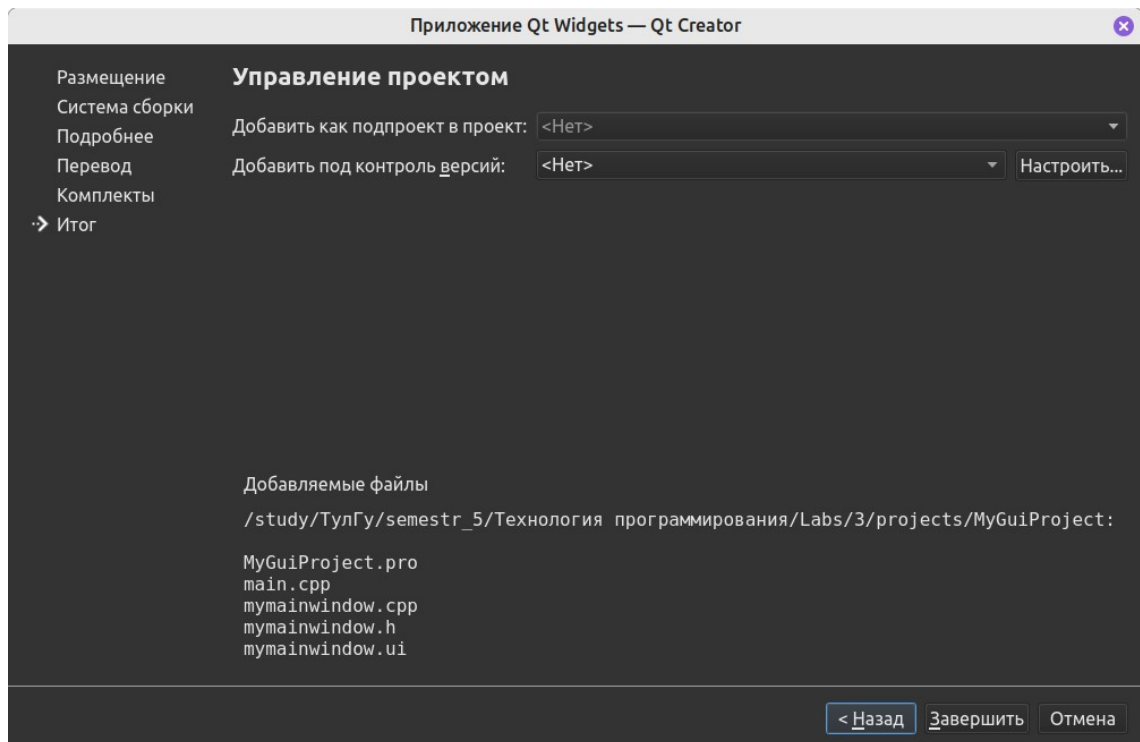
Вводим название класса главного окна `MyMainWindow`. Остальные параметры оставляем без изменения. Нажимаем кнопку **Далее >**.



Так как приложение создается в демонстрационных целях, то оставляем только комплект отладки. Нажимаем кнопку **Далее >**.

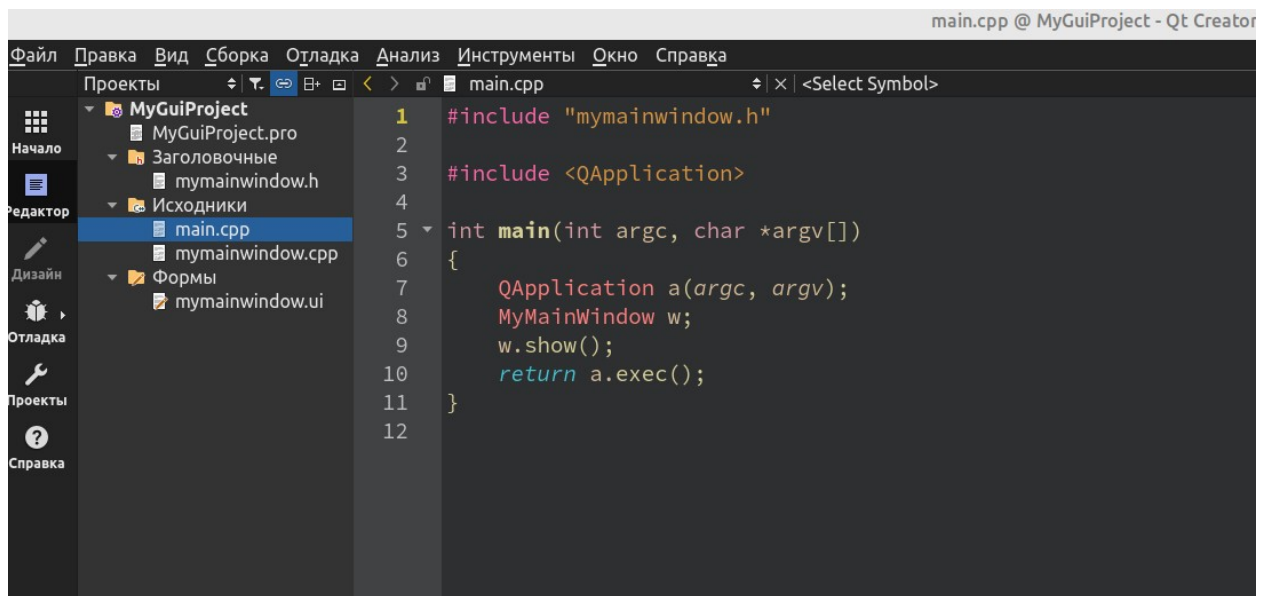


Далее оставляем все без изменений, так как данный проект не является подпроектом и мы не используем систему контроля версий. В нижней части окна выводится путь размещения проекта и список файлов, составляющих проект. Нажимаем кнопку Завершить.



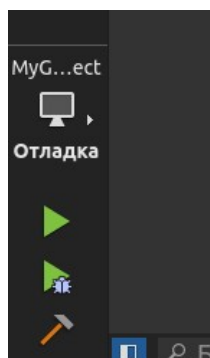
Проект имеет следующую структуру:

- файл `MyGuiProject.pro` — файл проекта, определяет основные параметры проекта, создается автоматически;
- файлы `mymainwindow.h` и `mymainwindow.cpp` — заголовочный файл и файл исходного кода класса `MyMainWindow`, создаются автоматически;
- файл `main.cpp` — файл исходного кода программы, является входной точкой в коде проекта, создается автоматически;
- файл `mymainwindow.ui` — файл формы класса `MyMainWindow`, содержит данные в формате `xml`, но через редактор `Qt Designer` осуществляется визуальное конструирование формы, создается автоматически.



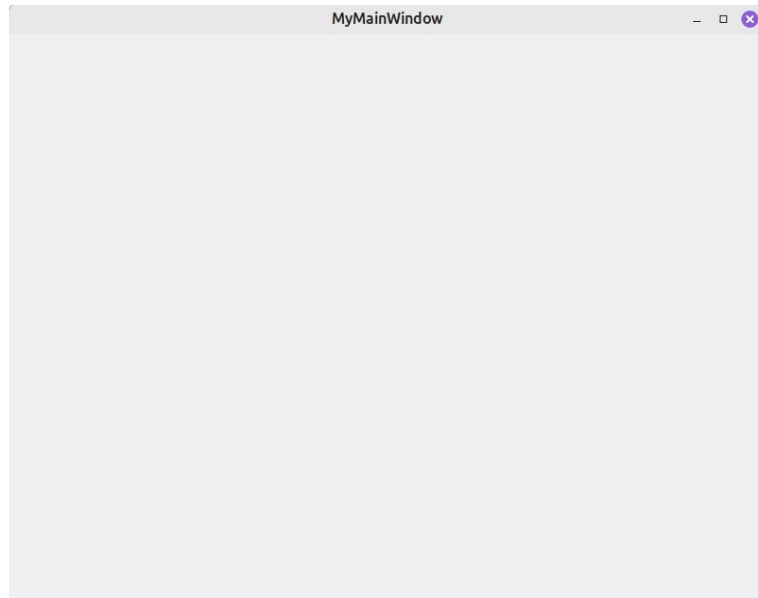
Компиляция и запуск проекта на выполнение выполняется несколькими способами:

- нажатие кнопки `Запустить` в левой нижней части редактора, в виде зеленого треугольника;



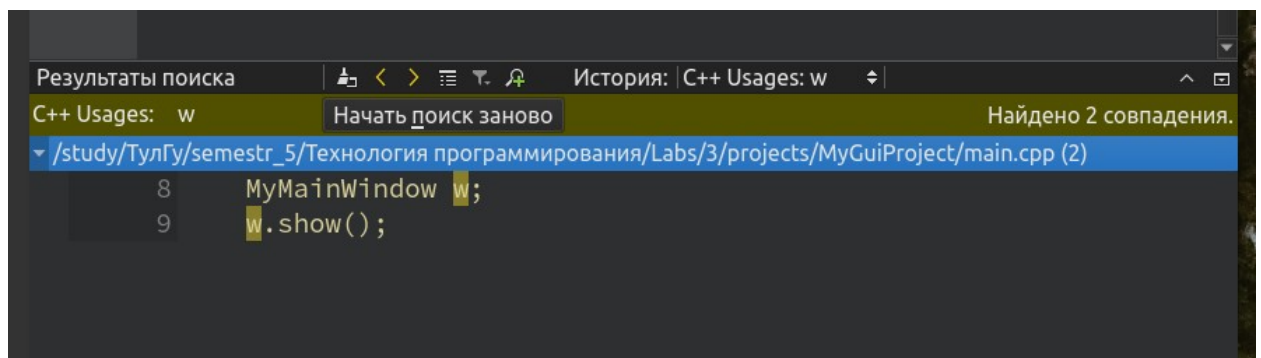
- комбинацией клавиш `Ctrl + R`;
- меню `Сборка` → `Запустить`.

В итоге проект компилируется без ошибок и запускается главное окно проекта с заголовком MyMainWindow. Окно совершенно пустое, так как по умолчанию не добавляется никаких элементов.



2. Как выше было указано компиляция и запуск проекта выполняется комбинацией клавиш Ctrl + R.

Так же полезной комбинацией является Ctrl+Shift+U, необходимой для поиска случаев использования переменной или класса в коде проекта.



Например, при наведении текстового курсора на переменную w в файле main.cpp и нажатии комбинации Ctrl+Shift+U появляется окно результатов поиска, в котором указаны имя файла и строки, которые используют переменную w.

3. В связи со сложившимися обстоятельствами компания Qt Group не предоставляет возможность пользователям из РФ скачивать установочные файлы Qt со своего сайта. Поэтому, мной было принято решение скачать исходные коды и выполнить компиляцию самостоятельно. После продолжительных танцев с бубном, я принял решение исключить документацию из компилируемого проекта, в виду значительной длительности компиляции.

За документацией возможно обратиться на сайт <https://doc.qt.io/qt-5/>.

Например, мануалы классов QMainWindow и QApplication оттуда:

Qt 5.15 > Qt Widgets > C++ Classes > QMainWindow

Q Search

QMainWindow Class

The QMainWindow class provides a main application window. [More...](#)

Header:	#include <QMainWindow>
qmake:	QT += widgets
Inherits:	QWidget

> List of all members, including inherited members

Public Types

enum	DockOption { AnimatedDocks, AllowNestedDocks, AllowTabbedDocks, ForceTabbedDocks, VerticalTabs, GroupedDragging }
flags	DockOptions

Properties

<div>> animated : bool</div> <div>> dockNestingEnabled : bool</div> <div>> dockOptions : DockOptions</div> <div>> documentMode : bool</div>	<div>> iconSize : QSize</div> <div>> tabShape : QTabWidget::TabShape</div> <div>> toolButtonStyle : Qt::ToolButtonStyle</div> <div>> unifiedTitleAndToolBarOnMac : bool</div>
---	---

Public Functions

	QMainWindow(QWidget *parent = nullptr, Qt::WindowFlags flags = Qt::WindowFlags())
virtual	~QMainWindow()
void	addDockWidget(Qt::DockWidgetArea area, QDockWidget *dockwidget)
void	addDockWidget(Qt::DockWidgetArea area, QDockWidget *dockwidget, Qt::Orientation orientation)
void	addToolBar(Qt::ToolBarArea area, QToolBar *toolbar)
void	addToolBar(QToolBar *toolbar)
QToolBar *	addToolBar(const QString &title)
void	addToolBarBreak(Qt::ToolBarArea area = Qt::TopToolBarArea)
QWidget *	centralWidget() const
Qt::DockWidgetArea	corner(Qt::Corner corner) const
virtual QMenu *	createPopupMenu()
QMainWindow::DockOptions	dockOptions() const
Qt::DockWidgetArea	dockWidgetArea(QDockWidget *dockwidget) const
bool	documentMode() const

QApplication Class

The QApplication class manages the GUI application's control flow and main settings. [More...](#)

Header:	<code>#include <QApplication></code>
qmake:	<code>QT += widgets</code>
Inherits:	QGuiApplication

- › [List of all members, including inherited members](#)
- › [Obsolete members](#)

Properties

› autoSipEnabled : bool	› startDragTime : int
› cursorFlashTime : int	› styleSheet : QString
› doubleClickInterval : int	› wheelScrollLines : int
› keyboardInputInterval : int	› windowIcon : QIcon
› startDragDistance : int	

Public Functions

	QApplication (int & <i>argc</i> , char ** <i>argv</i>)
virtual	~QApplication ()
QString	styleSheet () const

Reimplemented Public Functions

virtual bool	notify (QObject * <i>receiver</i> , QEvent * <i>e</i>) override
--------------	--

Public Slots

void	aboutQt ()
bool	autoSipEnabled () const
void	closeAllWindows ()
void	setAutoSipEnabled (const bool <i>enabled</i>)
void	setStyleSheet (const QString & <i>sheet</i>)

Signals

void	focusChanged (QWidget * <i>old</i> , QWidget * <i>now</i>)
------	---

Static Public Members

QWidget *	activeModalWidget ()
QWidget *	activePopupWidget ()
QWidget *	activeWindow ()

Лабораторная работа № 4.

Название работы: Структура проекта Qt. Классы для работы с текстовыми строками и файлами в Qt.

Цели работы: Изучение структуры Qt проекта и освоение принципов работы с основными классами.

Задачи:

1. Создайте пустой консольный проект Qt и скомпилируйте его. Определите потоковый оператор для вывода класса Person, который имеет поля Name, Phone number, Address для вывода в консоль с помощью функции qDebug().

2. Создайте программу табуляции функции и записи в текстовый файл с использованием средств, предоставляемых Qt. Используйте классы QFile, QTextStream для записи. Адрес текстового файла жёстко задайте в тексте программы.

3. Создайте программу чтения значений протабулированной функции из текстового файла с использованием средств Qt и вывода значений в консоль. Используйте классы QFile, QTextStream для чтения. Адрес текстового файла жёстко задайте в тексте программы. Если файл невозможно открыть для чтения – выведите сообщение о критической ошибке с завершением работы программы.

1. В запущенной среде Qt Creator открываем меню Файл → Создать файл или проект... Выбираем Приложение (Qt) → Консольное приложение Qt. Указываем размещение проекта и название Person.

Таким образом файл проекта Person.pro создается автоматически и имеет следующее содержимое:

```
QT -= gui

CONFIG += c++11 console
CONFIG -= app_bundle

# You can make your code fail to compile if it uses deprecated APIs.
# In order to do so, uncomment the following line.
#DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000
# disables all the APIs deprecated before Qt 6.0.0

SOURCES += \
    main.cpp

# Default rules for deployment.
qnx: target.path = /tmp/${TARGET}/bin
else: unix:!android: target.path = /opt/${TARGET}/bin
!isEmpty(target.path): INSTALLS += target
```

В данном коде видно, что из проекта исключен модуль gui и добавлен модуль console, используется стандарт c++11. Так же автоматически создан и добавлен в проект файл исходного кода main.cpp.

В файл main.cpp добавлен следующий код:

```
#include <QString>
#include <QDebug>

struct Person
{
    Person(const QString &name,
           const QString &phoneNumber,
           const QString &address) {
        this->name = name;
        this->phoneNumber = phoneNumber;
        this->address = address;
    }

    QString name;
    QString phoneNumber;
    QString address;
};

QDebug operator << (QDebug debug, const Person &person) {
    debug.nospace().noquote() << "Имя: " << person.name << " (телефон: "
        << person.phoneNumber << ", адрес: " << person.address << ")";
    return debug;
}

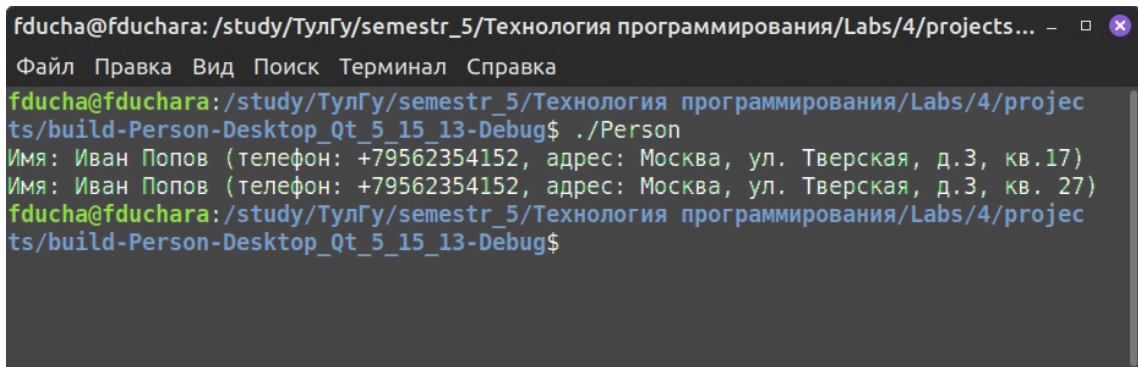
int main()
{
    Person p("Иван Попов", "+79562354152",
            "Москва, ул. Тверская, д.3, кв.17");
    qDebug() << p;
    p.address = "Москва, ул. Тверская, д.3, кв. 27";
    qDebug() << p;
    return 0;
}
```

В файл подключены заголовочные файлы классов QString и QDebug. QString — это собственный класс строки, реализованный в Qt. Класс QDebug используется для вывода сообщений в консоль в режиме отладки.

Класс Person реализован как структура и имеет открытые поля name, phoneNumber и address типа QString.

Оператор вывода для объектов класса Person реализован отдельной перегруженной функцией, принимающей поток вывода класса QDebug и константную ссылку на объект класса Person. При выводе отключены пробелы и кавычки.

При выполнении программы создается объект p класса Person с указанием имени, номера телефона и адреса. Объект p выводится при помощи функции qDebug(). Далее у объекта p изменяется адрес и объект снова выводится.



```
fducha@fduchara: /study/ТулГу/semestr_5/Технология программирования/Labs/4/projects... - □ ×
Файл Правка Вид Поиск Терминал Справка
fducha@fduchara: /study/ТулГу/semestr_5/Технология программирования/Labs/4/projects/build-Person-Desktop_Qt_5_15_13-Debug$ ./Person
Имя: Иван Попов (телефон: +79562354152, адрес: Москва, ул. Тверская, д.3, кв.17)
Имя: Иван Попов (телефон: +79562354152, адрес: Москва, ул. Тверская, д.3, кв. 27)
fducha@fduchara: /study/ТулГу/semestr_5/Технология программирования/Labs/4/projects/build-Person-Desktop_Qt_5_15_13-Debug$
```

2. В запущенной среде Qt Creator открываем меню Файл → Создать файл или проект... Выбираем Приложение (Qt) → Консольное приложение Qt. Указываем размещение проекта и название WriteFile.

Данная программа записывает в текстовый файл 10 строк, состоящих из числа от 1 до 10, знака подчеркивания и квадратного корня числа в начале строки, с точностью до 4 знаков после запятой.

Исходный код проекта:

```
#include <QFile>
#include <QTextStream>
#include <QIODevice>
#include <QDebug>
#include <math.h>

int main()
{
    QFile file("/study/ТулГу/semestr_5/Технология программирования/
               Labs/4/projects/data.txt");
    QTextStream in(&file);

    if (file.open(QIODevice::WriteOnly)) {
        for (int i = 1; i <= 10; i++) {
            QString result = QString("%1_%2\n").arg(i)
                               .arg(QString::number(sqrt(i), 'f', 4));
            in << qPrintable(result);
        }
        file.close();
    } else {
        qDebug() << "Error: Не могу открыть файл, прости";
    }

    return 0;
}
```

При выполнении программы создается объект file класса QFile. На основе данного файла создается поток in класса QTextStream. Далее файл file открывается на запись и в цикле от 1 до 10 создается строка формата «i_sqrt(i)» и отправляется в поток in. Файл закрывается. При ошибке открытия файла выводится сообщение об ошибке.

Снимок экрана вывода программы не приводится, так как программа не выводит никакой информации, а только создает файл.

3. В запущенной среде Qt Creator открываем меню Файл → Создать файл или проект... Выбираем Приложение (Qt) → Консольное приложение Qt. Указываем размещение проекта и название ReadFile.

Данная программа читает указанный текстовый файл, записанный в формате из предыдущего задания, преобразует полученные значения и выводит их на экран.

Исходный код проекта:

```
#include <QFile>
#include <QTextStream>
#include <QIODevice>
#include <QDebug>

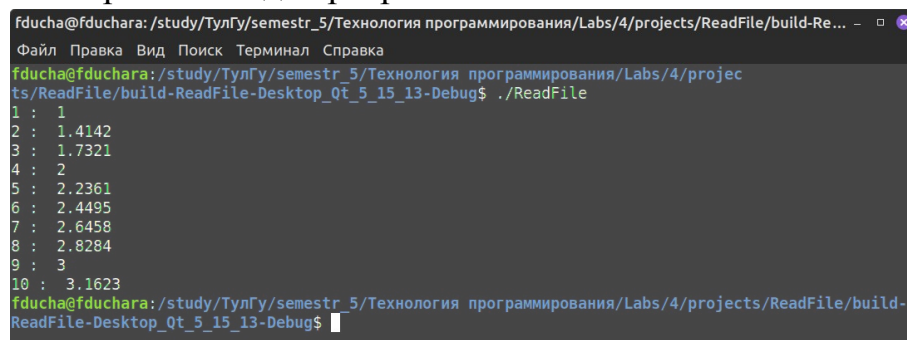
int main()
{
    QFile file("/study/ТулГу/semestr_5/Технология программирования
               /Labs/4/projects/data.txt");
    QTextStream in(&file);

    if (file.open(QIODevice::ReadOnly)) {
        while (!in.atEnd()) {
            QString raw = in.readLine();
            QStringList results = raw.split("_");
            int number = results[0].toInt();
            float data = results[1].toFloat();
            qDebug() << number << ": " << data;
        }
        file.close();
    } else {
        qCritical() << "Не могу открыть файл, прости";
        return 1;
    }

    return 0;
}
```

При выполнении программы создается объект file класса QFile. На основе данного файла создается поток in класса QTextStream. Далее файл file открывается на чтение и в цикле, пока не будет достигнут конец файла, читается построчно. Из каждой полученной строки создается список из 2 строк с разделением по символу «_». Первая строка приводится к целому и записывается в переменную number, вторая — приводится к типу float и записывается в переменную data. Переменные number и data выводятся в консоль через функцию qDebug(). Файл закрывается. При ошибке открытия файла выводится сообщение об ошибке.

Снимок экрана вывода программы:



```
fducha@fduchara: /study/ТулГу/semestr_5/Технология программирования/Labs/4/projects/ReadFile/build-Re...
fducha@fduchara: /study/ТулГу/semestr_5/Технология программирования/Labs/4/projects/ReadFile/build-ReadFile-Desktop_Qt_5_15_13-Debug$ ./ReadFile
1 : 1
2 : 1.4142
3 : 1.7321
4 : 2
5 : 2.2361
6 : 2.4495
7 : 2.6458
8 : 2.8284
9 : 3
10 : 3.1623
fducha@fduchara: /study/ТулГу/semestr_5/Технология программирования/Labs/4/projects/ReadFile/build-ReadFile-Desktop_Qt_5_15_13-Debug$
```

Лабораторная работа № 5.

Название работы: Создание графического интерфейса средствами Qt.

Цели работы: Изучение принципов создания графического интерфейса средствами Qt.

Задачи:

1. Используя IDE Qt Creator, ввести и отладить программы представленные в теоретическом материале.
2. Создайте проект с графическим интерфейсом. Разместите на окне в компоновщиках 5 различных виджетов. Соедините их (по 2–3 между собой) сигнально-слотовыми соединениями, таким образом, чтобы они реагировали на изменения состояния друг друга. (Например, чтобы QScrollBar реагировал на перемещение QSlider).

1. Примеры из теоретического материала

Widget

В запущенной среде Qt Creator открываем меню Файл → Создать файл или проект... Выбираем Другой проект → Пустой проект qmake. Указываем размещение проекта и название Lab5.

Добавляем следующее содержимое в про-файл:

```
TEMPLATE = app
QT += widgets
TARGET = widget
```

В левой панели проекты нажимаем правой кнопкой мыши на названии проекта, выбираем пункт Добавить новый ... Выбираем шаблон C/C++ → Файл исходных текстов C/C++, нажимаем кнопку Choose ... Вводим имя файла main.cpp, путь не трогаем, нажимаем кнопку Далее и в следующем окне Завершить. Таким образом, мы создали файл для исходного кода проекта, который автоматически добавлен в про-файл проекта:

```
TEMPLATE = app
QT += widgets
TARGET = widget
SOURCES += main.cpp
```

Вводим код примера в файл main.cpp:

```
#include <QApplication>
#include <QLabel>
#include <QFont>

int main(int argc, char *argv[]) {
    QApplication app(argc, argv);
```

```

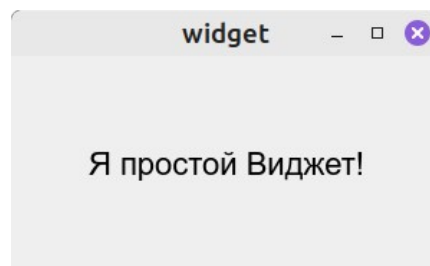
        QLabel label;
        label.setText("Я простой Виджет!");
        label.setGeometry(200, 200, 300, 150);
        label.setAlignment(Qt::AlignHCenter | Qt::AlignVCenter);
        QFont blackFont("Arial", 12);
        label.setFont(blackFont);
        label.show();

        return app.exec();
    }

```

В коде программы создается объект приложения `app` класса `QApplication` и надпись `label` класса `QLabel`. Надписи задается текст, размер, выравнивание в окне приложения и шрифт. Надпись делается видимой вызовом метода `show()` и запускается приложение `app`.

Запускаем проект нажатием `Ctrl-R` и видим окно приложения:



ParentExample

Аналогично предыдущему примеру создаем пустой проект, задаем ему название `ParentExample`.

Добавляем следующее содержимое в `pro`-файл проекта:

```

TEMPLATE = app
TARGET = ParentExample
QT += widgets

```

Аналогично предыдущему примеру создаем файл `main.cpp`.

Добавляем в проект собственный класс: в левой панели проекты нажимаем правой кнопкой мыши на названии проекта, выбираем пункт `Добавить новый ...`. Выбираем шаблон `C/C++ → Класс C++`, нажимаем кнопку `Choose ...`. Вводим имя класса `ParentExample`, базовый класс выбираем `QWidget`. ставим галочку `Подключить QWidget`, имена заголовочного файла, файла исходного кода и путь не трогаем, нажимаем кнопку `Далее` и в следующем окне `Завершить`. Таким образом, мы создали файлы класса `ParentExample`, которые автоматически добавлены в `pro`-файл проекта:

```

TEMPLATE = app
TARGET = ParentExample
QT += widgets

HEADERS += \
    parentwidget.h

```

```
SOURCES += \  
    main.cpp \  
    parentwidget.cpp
```

Заголовочный файл класса ParentExample оставляем без изменений:

```
#ifndef PARENTWIDGET_H  
#define PARENTWIDGET_H  
  
#include <QWidget>  
  
class ParentWidget : public QWidget  
{  
public:  
    explicit ParentWidget(QWidget *parent = nullptr);  
  
signals:  
  
};  
#endif // PARENTWIDGET_H
```

В файл исходного кода класса ParentExample вводим код примера:

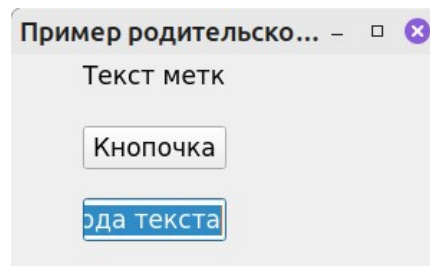
```
#include "parentwidget.h"  
#include <QLabel>  
#include <QPushButton>  
#include <QLineEdit>  
  
ParentWidget::ParentWidget(QWidget *parent)  
    : QWidget{parent}  
{  
    auto label = new QLabel(this);  
    label->setGeometry(50,0,100,30);  
    label->setText("Текст метки");  
    auto button = new QPushButton(this);  
    button->setGeometry(50,50,100,30);  
    button->setText("Кнопочка");  
    auto lineEdit = new QLineEdit(this);  
    lineEdit->setGeometry(50,100,100,30);  
    lineEdit->setText("Поле ввода текста");  
    lineEdit->selectAll();  
    setGeometry(x(), y(), 300, 150);  
    setWindowTitle("Пример родительского виджета");  
}
```

В данном коде подключены заголовочный файл класса ParentExample, хедеры классов QLabel, QPushButton, QLineEdit. Создаются виджеты надпись, кнопка и поле ввода. Виджетам при создании назначается родительский виджет ParentExample, далее задаются размеры и текст, устанавливается надпись на заголовке.

```
#include <QApplication>  
#include "parentwidget.h"  
  
int main(int argc, char *argv[]) {  
    QApplication app(argc, argv);  
    ParentWidget parent;  
    parent.move(100, 200);  
    parent.show();  
    return app.exec();  
}
```

В файле main.cpp подключен заголовочный файл класса ParentExample, создается виджет данного класса, виджет перемещен в позицию экрана с координатами (100, 200) и отображается, запускается приложение.

При запуске приложения отображается окно:



LayoutExample

В запущенной среде Qt Creator открываем меню Файл → Создать файл или проект... Выбираем Приложение (Qt) → Приложение Qt Widgets, нажимаем кнопку Choose... Указываем размещение проекта, задаем ему название LayoutExample, нажимаем Далее, Далее. Класс виджета главного окна приложения называем Widget, выбираем базовый класс QWidget, снимаем галочку с опции Создать форму, нажимаем 3 раза Далее, затем Завершить. Pro-файл проекта, файл main.cpp и файлы класса Widget сгенерированы автоматически.

Pro-файл проекта:

```
QT += core gui

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

CONFIG += c++11

SOURCES += \
    main.cpp \
    widget.cpp

HEADERS += \
    widget.h
# Default rules for deployment.
qnx: target.path = /tmp/${TARGET}/bin
else: unix:!android: target.path = /opt/${TARGET}/bin
!isEmpty(target.path): INSTALLS += target
```

Файл main.cpp:

```
#include "widget.h"

#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    Widget w;
    w.show();
    return a.exec();
}
```


Заголовочный файл класса Widget:

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>

class Widget : public QWidget
{
    Q_OBJECT

public:
    Widget(QWidget *parent = nullptr);
    ~Widget();
};
#endif // WIDGET_H
```

В файл исходного кода класса добавляем следующее:

```
#include "widget.h"
#include <QHBoxLayout>
#include <QVBoxLayout>
#include <QLabel>
#include <QLineEdit>
#include <QPushButton>

Widget::Widget(QWidget *parent)
    : QWidget(parent) {
    auto lineEdit = new QLineEdit("Text 1");
    auto label = new QLabel("Line Edit &1");
    label->setBuddy(lineEdit);
    auto hbLayout = new QHBoxLayout;
    hbLayout->addWidget(label);
    hbLayout->addWidget(lineEdit);

    auto lineEdit2 = new QLineEdit("Text 2");
    auto label2 = new QLabel("Line Edit &2");
    label2->setBuddy(lineEdit2);
    auto hbLayout2 = new QHBoxLayout;
    hbLayout2->addWidget(label2);
    hbLayout2->addWidget(lineEdit2);

    auto btnOK = new QPushButton("&OK");
    auto btnCancel = new QPushButton("&Cancel");
    auto hbLayout3 = new QHBoxLayout;

    hbLayout3->addStretch();
    hbLayout3->addWidget(btnOK);
    hbLayout3->addWidget(btnCancel);

    auto vbLayout = new QVBoxLayout;
    vbLayout->addLayout(hbLayout);
    vbLayout->addLayout(hbLayout2);
    vbLayout->addLayout(hbLayout3);

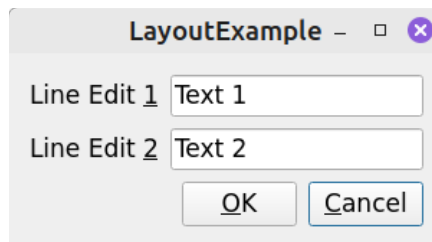
    setLayout(vbLayout);
}

Widget::~~Widget() {}
```

В конструкторе виджета создаются попарно 2 группы виджетов надписи и поля ввода, которые компоуются в горизонтальные макеты разметки. Далее создается группа кнопок, тоже скомпонованная в

горизонтальный макет разметки, но с добавлением виджета-растяжителя. Далее создается вертикальный макет разметки, который задается как базовый макет виджета, и в который добавляются уже созданные горизонтальные макеты.

При запуске приложения отображается окно:



Calculator

Создаем проект Calculator аналогично предыдущего примера, виджет главного окна называем CalculatorMainWindow. Pro-файл проекта, файл main.cpp и файлы класса CalculatorMainWindow генерируются автоматически.

Файл main.cpp:

```
#include "calculatormainwindow.h"

#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    CalculatorMainWindow w;
    w.show();
    return a.exec();
}
```

Заголовочный файл класса CalculatorMainWindow:

```
#ifndef CALCULATORMAINWINDOW_H
#define CALCULATORMAINWINDOW_H

#include <QWidget>

class QPushButton;
class QLCDNumber;
class QSignalMapper;

class CalculatorMainWindow : public QWidget
{
    Q_OBJECT

public:
    CalculatorMainWindow(QWidget *parent = nullptr);
    ~CalculatorMainWindow();

private slots:
    void clear();
    void btnPressed(int num);
    void plusEqualPressed();
}
```

```

private:
    void createWidgets();

    QPushButton *m_btn0;
    QPushButton *m_btn1;
    QPushButton *m_btn2;
    QPushButton *m_btn3;
    QPushButton *m_btn4;
    QPushButton *m_btn5;
    QPushButton *m_btn6;
    QPushButton *m_btn7;
    QPushButton *m_btn8;
    QPushButton *m_btn9;
    QPushButton *m_btnPlus;
    QPushButton *m_btnC;

    QLCDNumber *LCDDisplay;

    QSignalMapper *m_mapper;

    int m_sum;
    int m_nextNumber;
};
#endif // CALCULATORMAINWINDOW_H

```

В заголовочный файл добавлено предварительное объявление классов кнопки QPushButton, дисплея QLCDNumber и маппера QSignalMapper. Добавлены приватные слоты:

clear() - очистка дисплея и сброс вычислений;
 btnPressed(int num) — обработка нажатия цифровых кнопок;
 plusEqualPressed() — обработка нажатия кнопки Сложить/Равно.

Закрытый метод createWidgets() выполняет создание виджетов и размещает их в макете класса QGridLayout. Так же объявляются указатели на цифровые кнопки, кнопки сложения и сброса, дисплея и маппера. Объявляются целые переменные для хранения введенного числа и результата вычислений.

Исходный код класса CalculatorMainWindow:

```

#include "calculatormainwindow.h"
#include <QPushButton>
#include <QGridLayout>
#include <QLCDNumber>
#include <QSignalMapper>

CalculatorMainWindow::CalculatorMainWindow(QWidget *parent)
    : QWidget(parent)
{
    resize(300, 300);
    setWindowTitle("Простой калькулятор");
    createWidgets();

    connect(m_btnC, &QPushButton::clicked,
            this, &CalculatorMainWindow::clear, Qt::UniqueConnection);
    connect(m_btnPlus, &QPushButton::clicked,
            this, &CalculatorMainWindow::plusEqualPressed, Qt::UniqueConnection);

    m_mapper = new QSignalMapper(this);
    connect(m_btn0, SIGNAL(clicked()), m_mapper, SLOT(map()),
            Qt::UniqueConnection);
    connect(m_btn1, SIGNAL(clicked()), m_mapper, SLOT(map()),

```

```

        Qt::UniqueConnection);
connect(m_btn2, SIGNAL(clicked()), m_mapper, SLOT(map()),
        Qt::UniqueConnection);
connect(m_btn3, SIGNAL(clicked()), m_mapper, SLOT(map()),
        Qt::UniqueConnection);
connect(m_btn4, SIGNAL(clicked()), m_mapper, SLOT(map()),
        Qt::UniqueConnection);
connect(m_btn5, SIGNAL(clicked()), m_mapper, SLOT(map()),
        Qt::UniqueConnection);
connect(m_btn6, SIGNAL(clicked()), m_mapper, SLOT(map()),
        Qt::UniqueConnection);
connect(m_btn7, SIGNAL(clicked()), m_mapper, SLOT(map()),
        Qt::UniqueConnection);
connect(m_btn8, SIGNAL(clicked()), m_mapper, SLOT(map()),
        Qt::UniqueConnection);
connect(m_btn9, SIGNAL(clicked()), m_mapper, SLOT(map()),
        Qt::UniqueConnection);

m_mapper->setMapping(m_btn0, 0);
m_mapper->setMapping(m_btn1, 1);
m_mapper->setMapping(m_btn2, 2);
m_mapper->setMapping(m_btn3, 3);
m_mapper->setMapping(m_btn4, 4);
m_mapper->setMapping(m_btn5, 5);
m_mapper->setMapping(m_btn6, 6);
m_mapper->setMapping(m_btn7, 7);
m_mapper->setMapping(m_btn8, 8);
m_mapper->setMapping(m_btn9, 9);

clear();

connect(m_mapper, SIGNAL(mapped(int)), this, SLOT(btnPressed(int)),
        Qt::UniqueConnection);
}

CalculatorMainWindow::~CalculatorMainWindow()
{
}

void CalculatorMainWindow::clear() {
    LCDisplay->display(0);
    m_sum = 0;
    m_nextNumber = 0;
}

void CalculatorMainWindow::btnPressed(int num) {
    m_nextNumber = num;
    LCDisplay->display(num);
}

void CalculatorMainWindow::plusEqualPressed() {
    m_sum += m_nextNumber;
    LCDisplay->display(m_sum);
    m_nextNumber = 0;
}

void CalculatorMainWindow::createWidgets() {
    auto lytCalc = new QGridLayout;
    setLayout(lytCalc);

    LCDisplay = new QLCDNumber;

    m_btn0 = new QPushButton(" 0 ");
    m_btn1 = new QPushButton(" 1 ");
    m_btn2 = new QPushButton(" 2 ");
    m_btn3 = new QPushButton(" 3 ");

```

```

m_btn4 = new QPushButton(" 4 ");
m_btn5 = new QPushButton(" 5 ");
m_btn6 = new QPushButton(" 6 ");
m_btn7 = new QPushButton(" 7 ");
m_btn8 = new QPushButton(" 8 ");
m_btn9 = new QPushButton(" 9 ");
m_btnPlus = new QPushButton(" +/= ");
m_btnC = new QPushButton(" C ");

lytCalc->addWidget(LCDisplay, 0, 0, 1, 4);
lytCalc->addWidget(m_btn1, 1, 0);
lytCalc->addWidget(m_btn2, 1, 1);
lytCalc->addWidget(m_btn3, 1, 2);
lytCalc->addWidget(m_btn4, 2, 0);
lytCalc->addWidget(m_btn5, 2, 1);
lytCalc->addWidget(m_btn6, 2, 2);
lytCalc->addWidget(m_btn7, 3, 0);
lytCalc->addWidget(m_btn8, 3, 1);
lytCalc->addWidget(m_btn9, 3, 2);
lytCalc->addWidget(m_btn0, 4, 0, 1, 3);
lytCalc->addWidget(m_btnC, 1, 3);
lytCalc->addWidget(m_btnPlus, 2, 3, 3, 1);

m_btn0->setSizePolicy(QSizePolicy::Preferred, QSizePolicy::Preferred);
m_btn1->setSizePolicy(QSizePolicy::Preferred, QSizePolicy::Preferred);
m_btn2->setSizePolicy(QSizePolicy::Preferred, QSizePolicy::Preferred);
m_btn3->setSizePolicy(QSizePolicy::Preferred, QSizePolicy::Preferred);
m_btn4->setSizePolicy(QSizePolicy::Preferred, QSizePolicy::Preferred);
m_btn5->setSizePolicy(QSizePolicy::Preferred, QSizePolicy::Preferred);
m_btn6->setSizePolicy(QSizePolicy::Preferred, QSizePolicy::Preferred);
m_btn7->setSizePolicy(QSizePolicy::Preferred, QSizePolicy::Preferred);
m_btn8->setSizePolicy(QSizePolicy::Preferred, QSizePolicy::Preferred);
m_btn9->setSizePolicy(QSizePolicy::Preferred, QSizePolicy::Preferred);
m_btn9->setSizePolicy(QSizePolicy::Preferred, QSizePolicy::Preferred);
m_btnPlus->setSizePolicy(QSizePolicy::Preferred, QSizePolicy::Preferred);
m_btnC->setSizePolicy(QSizePolicy::Preferred, QSizePolicy::Preferred);

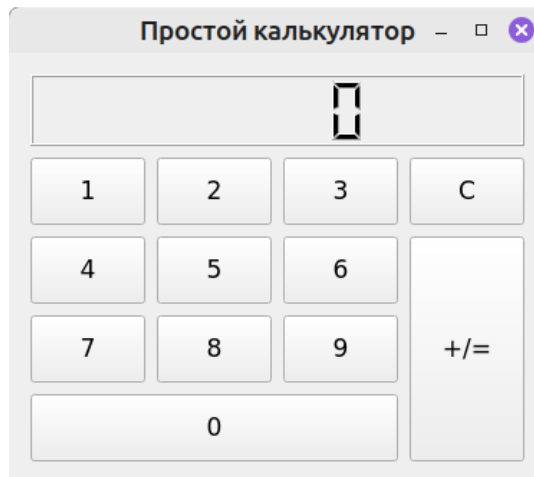
LCDDisplay->setFixedHeight(50);
}

```

В конструкторе класса задаются размеры окна программы и текст заголовка, вызывается метод `createWidgets()` для создания виджетов, выполняется соединение слотов `clear()` и `plusEqualPressed()` с сигналами нажатия кнопок сброса и сложения соответственно. Далее выполняется настройка маппинга нажатия цифровых кнопок: сигналы нажатия цифровых кнопок соединяются со слотом `map()` маппера, мапперу задаются ассоциации каждой цифровой кнопки с индексом, (например, кнопке 0 задается индекс 0, кнопке 1 — индекс 1 и так далее), и выполняется соединение сигнала маппера со слотом обработки нажатия цифровых кнопок `btnPressed(int num)`. Работает это следующим образом: нажимается цифровая кнопка, сигнал об этом передается мапперу, маппер определяет с каким индексом связана данная кнопка и передает этот индекс посредством сигнала `mapped(int)` обработчику нажатия кнопок `btnPressed(int num)`.

Слот `btnPressed(int num)` выводит значение нажатой кнопки на дисплей, а слот `plusEqualPressed()` выполняет сложение введенного значения с предыдущим введенным значением.

При запуске приложения отображается окно:



2. Проект с графическим интерфейсом

Создаем проект Signals аналогично предыдущим примерам, но не отключаем генерацию файла формы signalswidget.ui, виджет главного окна называем SignalsWidget. Pro-файл проекта, файл main.cpp и файлы класса SignalsWidget генерируются автоматически.

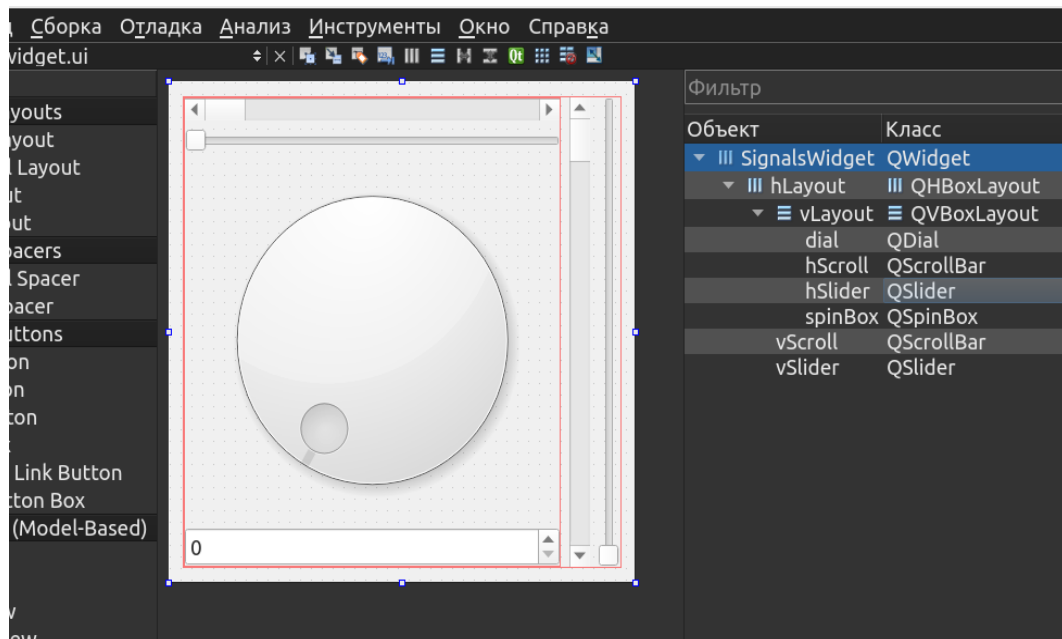
Файл main.cpp:

```
#include "signalswidget.h"

#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    SignalsWidget w;
    w.show();
    return a.exec();
}
```

Для оформления формы виджета дважды кликаем на файле формы signalswidget.ui в левой боковой панели Проекты в подразделе Формы, выполняется переход в раздел Дизайн. Добавляем на форму горизонтальные виджеты QSlider, QScrollBar, виджеты QDial и QSpinBox. Компонуем их в вертикальный макет — выделяем их и нажимаем кнопку Скомпоновать по вертикали (Ctrl+L). Добавляем вертикальные виджеты QSlider и QScrollBar. Компонуем вертикальные виджеты и макет с вертикально скомпонованными виджетами по горизонтали (кнопка Скомпоновать по горизонтали или Ctrl+H). Получается следующая форма:



Заголовочный файл класса SignalsWidget:

```
#ifndef SIGNALSWIDGET_H
#define SIGNALSWIDGET_H

#include <QWidget>

QT_BEGIN_NAMESPACE
namespace Ui { class SignalsWidget; }
QT_END_NAMESPACE

class SignalsWidget : public QWidget
{
    Q_OBJECT

public:
    SignalsWidget(QWidget *parent = nullptr);
    ~SignalsWidget();

private slots:
    void valueChanged(int value);
private:
    Ui::SignalsWidget *ui;
};
#endif // SIGNALSWIDGET_H
```

В коде добавлен приватный слот valueChanged(int value) для изменения значения положения ползунков виджетов.

Файл исходного кода класса SignalsWidget:

```
#include "signalswidget.h"
#include "ui_signalswidget.h"

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

    connect(ui->dial, SIGNAL/sliderMoved(int)), this,
```

```

        SLOT(valueChanged(int)));
connect(ui->hScroll, SIGNAL(sliderMoved(int)), this,
        SLOT(valueChanged(int)));
connect(ui->hSlider, SIGNAL(sliderMoved(int)), this,
        SLOT(valueChanged(int)));
connect(ui->vScroll, SIGNAL(sliderMoved(int)), this,
        SLOT(valueChanged(int)));
connect(ui->vSlider, SIGNAL(sliderMoved(int)), this,
        SLOT(valueChanged(int)));
connect(ui->spinBox, SIGNAL(valueChanged(int)), this,
        SLOT(valueChanged(int)));
}

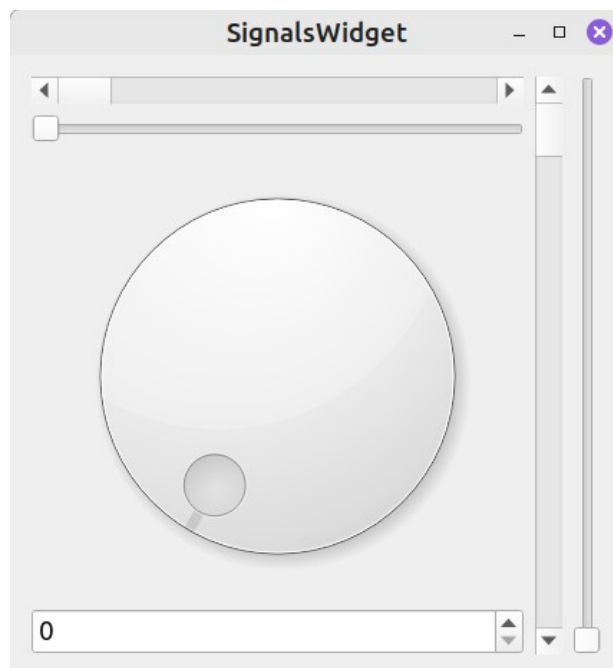
SignalsWidget::~SignalsWidget()
{
    delete ui;
}

void SignalsWidget::valueChanged(int value) {
    ui->dial->setValue(value);
    ui->hScroll->setValue(value);
    ui->hSlider->setValue(value);
    ui->vScroll->setValue(value);
    ui->vSlider->setValue(value);
    ui->spinBox->setValue(value);
}

```

В конструкторе виджета выполняем соединение сигналов виджетов об изменении положения ползунков со слотом `valueChanged(int value)`, который изменяет положение ползунков остальных виджетов на переданное значение.

При запуске приложения отображается окно:



В файле `lab5.mkv` содержится видео демонстрации работы программы.

Лабораторная работа № 6.

Название работы: Собственные классы в Qt. Создание элементов графического интерфейса.

Цели работы: Изучение принципов построения и создания классов в Qt.

Задачи:

1. Создайте окно для ввода имени пользователя и пароля. Используйте для этого класс `IconizedLineEdit`.
2. Добавьте возможность проверки правильности введённых данных с помощью класса `QValidator`.
3. Создайте программу, которая открывает изображение в окне. Для ввода пути к изображению добавьте к окну поле ввода. Для поля ввода пути к изображению используйте класс `IconizedLineEdit`. При нажатии на пиктограмму должен открываться диалог выбора файла.
4. Добавьте к классу `LedIndicator` поддержку третьего промежуточного состояния. Для включения поддержки этого состояния, добавьте метод `setTristate()`. Для установки и чтения состояния индикатора добавьте методы `setState`.

Перед началом работы с заданиями созданы примеры проектов `IconizedLineEdit` и `LedIndicator` из теоретической части работы. Примеры отлажены и работоспособны.

Код класса `IconizedLineEdit`.

Заголовочный файл `iconizedlineedit.h`:

```
#ifndef ICONIZEDLINEEDIT_H
#define ICONIZEDLINEEDIT_H

#include <QLineEdit>
#include <QEvent>

class QLabel;

class IconizedLineEdit : public QLineEdit
{
    Q_OBJECT
public:
    enum IconVisibleMode {
        IconAlwaysVisible = 0,
        IconVisibleOnHover,
        IconVisibleOnTextPresence,
        IconVisibleOnEmptyText,
        IconAlwaysHidden,
    };

    explicit IconizedLineEdit(QWidget *parent = nullptr);

    void setIconVisiblity(IconVisibleMode mode);
```

```

        bool isVisible() const;
        void setIconPixmap(const QPixmap &pixmap);
        void setIconToolTip(const QString &toolTip);
        void setIconClickable(bool clickable);

signals:
    void iconPressed();

protected:
    virtual void resizeEvent(QResizeEvent *event) override;
    virtual bool eventFilter(QObject *obj, QEvent *event) override;

private slots:
    void textChanged(const QString &text);

private:
    void updateIconPositionAndSize();
    void setIconVisible(bool isVisible);

    QLabel *m_iconLabel;
    IconVisibleMode m_mode;
    bool m_isIconClickable;
};

#endif // ICONIZEDLINEEDIT_H

```

Исходный код iconizedlineedit.cpp:

```

#include "iconizedlineedit.h"
#include <QStyle>
#include <QLabel>

IconizedLineEdit::IconizedLineEdit(QWidget *parent)
    : QLineEdit{parent}, m_mode{IconAlwaysVisible}
{
    m_iconLabel = new QLabel(this);
    m_iconLabel->installEventFilter(this);
    connect(this, SIGNAL(textChanged(QString)),
            this, SLOT(textChanged(QString)), Qt::UniqueConnection);
}

void IconizedLineEdit::setIconVisiblity(IconVisibleMode mode) {
    m_mode = mode;

    switch (mode) {
    case IconAlwaysVisible:
        setIconVisible(true);
        break;
    case IconVisibleOnEmptyText:
    case IconVisibleOnTextPresence:
        textChanged(text());
        break;
    default:
        setIconVisible(false);
        break;
    }
}

bool IconizedLineEdit::isVisible() const {
    return m_iconLabel->isVisible();
}

void IconizedLineEdit::setIconPixmap(const QPixmap &pixmap) {
    m_iconLabel->setPixmap(pixmap.scaled(24, 24, Qt::KeepAspectRatio));
}

```

```

        m_iconLabel->setFixedSize(24, 24);
        updateIconPositionAndSize();
    }

void IconizedLineEdit::setIconToolTip(const QString &toolTip) {
    m_iconLabel->setToolTip(toolTip);
}

void IconizedLineEdit::setIconClickable(bool clickable) {
    m_isIconClickable = clickable;
    if (m_isIconClickable) m_iconLabel->setCursor(Qt::PointingHandCursor);
    else m_iconLabel->setCursor(Qt::ArrowCursor);
}

void IconizedLineEdit::resizeEvent(QResizeEvent *event) {
    updateIconPositionAndSize();
    QWidget::resizeEvent(event);
}

bool IconizedLineEdit::eventFilter(QObject *obj, QEvent *event) {
    if (m_isIconClickable) {
        if ((obj == m_iconLabel)
            && (event->type() == QEvent::MouseButtonPress)) {
            emit iconPressed();
            return true;
        }
    }
    return false;
}

void IconizedLineEdit::textChanged(const QString &text) {
    if (m_mode == IconVisibleOnEmptyText) {
        setIconVisible(text.isEmpty());
    } else if (m_mode == IconVisibleOnTextPresence) {
        setIconVisible(!text.isEmpty());
    }
}

void IconizedLineEdit::updateIconPositionAndSize() {
    if (m_iconLabel->isVisible()) {
        setTextMargins(QMargins(1, 1, 25, 1));
    } else {
        setTextMargins(QMargins(1, 1, 1, 1));
    }
    m_iconLabel->setScaledContents(true);
    auto size = m_iconLabel->size();
    m_iconLabel->move(rect().right() - size.width() - 2, rect().top() + 2);
}

void IconizedLineEdit::setIconVisible(bool isVisible) {
    m_iconLabel->setVisible(isVisible);
}

```

Код класса LedIndicator.

Заголовочный файл ledindicator.h:

```

#ifndef LEDINDICATOR_H
#define LEDINDICATOR_H

#include <QWidget>

class LedIndicator : public QWidget
{

```

```

    Q_OBJECT
    Q_PROPERTY(QString text READ text WRITE setText)
    Q_PROPERTY(bool turnedOn READ isTurnedOn WRITE setTurnedOn
        NOTIFY stateToggled)

public:
    explicit LedIndicator(QWidget *parent = nullptr);

    QString text() const;
    bool isTurnedOn() const;

    virtual QSize minimumSizeHint() const override;

signals:
    void stateToggled(bool);

public slots:
    void setText(const QString &text);
    void setTurnedOn(bool turned);

protected:
    virtual void paintEvent(QPaintEvent *event) override;

private:
    QString m_text;
    bool m_isTurnedOn;
};

#endif // LEDINDICATOR_H

```

Исходный код ledindicator.cpp:

```

#include "ledindicator.h"
#include <QPainter>
#include <QPainterPath>
#include <QPaintEvent>

const int LED_RADIUS = 15;
const int LED_SPACING = 5;

LedIndicator::LedIndicator(QWidget *parent)
    : QWidget{parent}, m_isTurnedOn{false}
{
}

QString LedIndicator::text() const {
    return m_text;
}

bool LedIndicator::isTurnedOn() const {
    return m_isTurnedOn;
}

QSize LedIndicator::minimumSizeHint() const {
    return QSize(LED_RADIUS * 2 + fontMetrics().horizontalAdvance(m_text) +
        LED_SPACING, LED_SPACING * 2);
}

void LedIndicator::setText(const QString &text) {
    m_text = text;
}

void LedIndicator::setTurnedOn(bool turned) {
    if (isTurnedOn() == turned) return;
}

```

```

        m_isTurnedOn = turned;
        emit stateToggled(m_isTurnedOn);
        update();
    }

void LedIndicator::paintEvent(QPaintEvent *event) {
    Q_UNUSED(event);

    QPainter painter(this);
    painter.setRenderHint(QPainter::Antialiasing);

    QPoint ledCenter(LED_RADIUS + 1, height() / 2);

    QPainterPath path;
    path.addEllipse(ledCenter, LED_RADIUS, LED_RADIUS);
    painter.save();

    QRadialGradient gradient(ledCenter, LED_RADIUS);

    if (m_isTurnedOn) {
        painter.setPen(QPen(Qt::darkGreen));
        gradient.setColorAt(0.2, "#70FF70");
        gradient.setColorAt(1, "#00CC00");
    } else {
        painter.setPen(QPen(Qt::black));
        gradient.setColorAt(0.2, Qt::gray);
        gradient.setColorAt(1, Qt::darkGray);
    }

    painter.fillPath(path, QBrush(gradient));
    painter.drawPath(path);

    painter.restore();
    painter.setFont(font());

    QRect textRect(LED_RADIUS * 2 + LED_SPACING, 0,
                    width() - (LED_RADIUS * 2 + LED_SPACING), height());
    painter.drawText(textRect, Qt::AlignVCenter | Qt::AlignLeft, m_text);
}

```

1. Окно для ввода имени пользователя и пароля.

В запущенной среде Qt Creator открываем меню Файл → Создать файл или проект... Выбираем Приложение (Qt) → Приложение Qt Widgets, нажимаем кнопку Choose... Указываем размещение проекта, задаем ему название LogIn, нажимаем Далее, Далее. Класс виджета главного окна приложения называем MainWindow, выбираем базовый класс QWidget, форму окна приложения не генерируем. Pro-файл проекта, файл main.cpp и файлы класса MainWindow сгенерированы автоматически.

Копируем в каталог исходного кода проекта файлы iconizedlineedit.h и iconizedlineedit.cpp из проекта IconizedLineEdit и добавляем их как существующие файлы.

Файл main.cpp полностью аналогичен предыдущим примерам, поэтому не рассматривается.

Заголовочный файл mainwindow.h:

```

#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QWidget>

class QPushButton;
class IconizedLineEdit;

class MainWindow : public QWidget
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private:
    void createUi();

private slots:
    void logged();

private:
    IconizedLineEdit *m_ileLogin;
    IconizedLineEdit *m_ilePassword;
    QPushButton *m_btnLogin;
    QPushButton *m_btnClose;
};
#endif // MAINWINDOW_H

```

В данный код внесено предварительное объявление классов QPushButton и IconizedLineEdit, объявление закрытого метода createUi() и слота logged(), указателей на поля ввода логина и пароля класса IconizedLineEdit и кнопок входа и закрытия окна класса QPushButton.

Файл исходного кода mainwindow.cpp:

```

#include "mainwindow.h"
#include <QVBoxLayout>
#include <QHBoxLayout>
#include <QPushButton>
#include <QMessageBox>
#include "iconizedlineedit.h"

MainWindow::MainWindow(QWidget *parent)
    : QWidget(parent)
{
    createUi();

    connect(m_btnClose, &QPushButton::clicked,
            this, &MainWindow::close);
    connect(m_btnLogin, &QPushButton::clicked,
            this, &MainWindow::logged);
}

MainWindow::~MainWindow() {}

void MainWindow::createUi() {
    auto lytMain = new QVBoxLayout;
    setLayout(lytMain);

    m_ileLogin = new IconizedLineEdit;

```

```

m_ileLogin->setPlaceholderText("Введите логин");
m_ileLogin->setIconVisiblity(IconizedLineEdit::IconVisibleOnEmptyText);
m_ileLogin->setIconPixmap(QPixmap("user.png"));
m_ileLogin->setIconClickable(false);
m_ileLogin->setMinimumWidth(300);
lytMain->addWidget(m_ileLogin);

m_ilePassword = new IconizedLineEdit(this);
m_ilePassword->setPlaceholderText("Введите пароль");
m_ilePassword->setIconVisiblity(IconizedLineEdit::IconVisibleOnEmptyText);
m_ilePassword->setIconPixmap(QPixmap("unlock.png"));
m_ilePassword->setIconClickable(false);
m_ilePassword->setEchoMode(QLineEdit::Password);
lytMain->addWidget(m_ilePassword);

auto lytButtons = new QHBoxLayout;

lytButtons->addStretch();
m_btnLogin = new QPushButton("&Вход");
lytButtons->addWidget(m_btnLogin);
m_btnClose = new QPushButton("&Заккрыть");
lytButtons->addWidget(m_btnClose);

lytMain->addLayout(lytButtons);
}

void MainWindow::logged() {
    if (m_ileLogin->text().isEmpty() || m_ilePassword->text().isEmpty())
        QMessageBox::information(nullptr, "Вход в систему",
                                   "Не введен логин или пароль");
    else {
        QMessageBox::information(nullptr, "Вход в систему",
                                   "Выполнен успешный вход в систему");
        close();
    }
}
}

```

В коде метода `createUi()` создается поле ввода логина класса `IconizedLineEdit`, полю устанавливаются замещающий текст, тип видимости иконки (отображается при отсутствии текста), сама иконка (не кликабельна) и минимальный размер, поле ввода логина добавляется в вертикальный компоновщик окна.

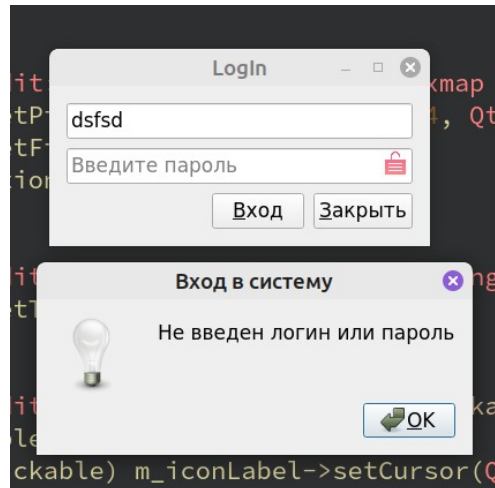
Далее создается поле ввода пароля класса `IconizedLineEdit`, полю устанавливаются замещающий текст, тип видимости иконки (отображается при отсутствии текста), иконка (не кликабельна) и устанавливается режим ввода пароля (введенные символы отображаются как *), поле ввода пароля добавляется в вертикальный компоновщик окна.

Далее создается горизонтальный компоновщик, в который добавлены растяжитель, кнопки входа и закрытия окна, компоновщик добавляется в компоновщик окна.

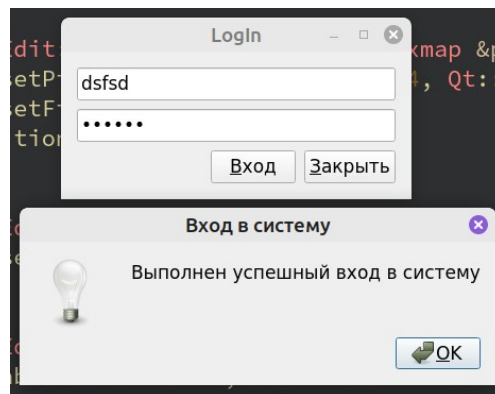
Метод `logged()` реализует заглушку входа: если поле ввода логина или пароля пустое, то отображается сообщение об этом, иначе отображается сообщение об успешном входе в систему.

В конструкторе класса выполнено соединение сигнала нажатия кнопки `Заккрыть` со слотом окна `close()`, а так же сигнала нажатия кнопки `Вход` со слотом окна `logged()`.

Снимок окна приложения при не введенном пароле:



Снимок окна приложения при заполненных полях логин и пароль:



2. Окно для ввода имени пользователя и пароля с проверкой правильности введенных данных.

Для проверки правильности введенных данных логина и пароля использован класс валидатора `QValidator`.

Логика проверки следующая: для логина возможно использовать слово на латинице длиной от 2 до 10 символов, причем первый символ — обязательно заглавный. Пароль — цифровой из 6 символов.

В файл `mainwindow.h` добавлено только объявление закрытого слота `onValidate()`.

Изменения, внесенные только в файл `mainwindow.cpp`:

```
#include "mainwindow.h"
#include <QVBoxLayout>
#include <QHBoxLayout>
#include <QPushButton>
#include <QValidator>
#include <QMessageBox>
#include "iconizedlineedit.h"

MainWindow::MainWindow(QWidget *parent)
    : QWidget(parent)
{
```



```

        createUi();
    }

MainWindow::~MainWindow()
{
}

void MainWindow::createUi() {
    auto lytMain = new QVBoxLayout;
    setLayout(lytMain);

    m_ileLogin = new IconizedLineEdit;
    m_ileLogin->setPlaceholderText("Введите логин");
    m_ileLogin->setIconVisiblity(IconizedLineEdit::IconVisibleOnEmptyText);
    m_ileLogin->setIconPixmap(QPixmap("user.png"));
    m_ileLogin->setIconClickable(false);
    m_ileLogin->setMinimumWidth(300);
    // Логин пользователя с заглавной буквы
    // длиной от 2 до 10 символов латинского алфавита
    m_ileLogin->setValidator(
        new QRegExpValidator(QRegExp("[A-Z][a-z]{1,9}"))
    );
    lytMain->addWidget(m_ileLogin);

    m_ilePassword = new IconizedLineEdit(this);
    m_ilePassword->setPlaceholderText("Введите пароль");
    m_ilePassword->setIconVisiblity(IconizedLineEdit::IconVisibleOnEmptyText);
    m_ilePassword->setIconPixmap(QPixmap("unlock.png"));
    m_ilePassword->setIconClickable(false);
    m_ilePassword->setEchoMode(QLineEdit::Password);
    // пароль пользователя пин-код длиной 6 цифр
    m_ilePassword->setValidator(new QRegExpValidator(QRegExp("[0-9]{6}")));
    lytMain->addWidget(m_ilePassword);

    connect(m_ileLogin, &IconizedLineEdit::textEdited,
        this, &MainWindow::onValidate);
    connect(m_ilePassword, &IconizedLineEdit::textEdited,
        this, &MainWindow::onValidate);

    auto lytButtons = new QHBoxLayout;

    lytButtons->addStretch();
    m_btnLogin = new QPushButton("&Вход");
    m_btnLogin->setEnabled(false);
    lytButtons->addWidget(m_btnLogin);
    m_btnClose = new QPushButton("&Заккрыть");
    lytButtons->addWidget(m_btnClose);

    lytMain->addLayout(lytButtons);

    connect(m_btnClose, &QPushButton::clicked,
        this, &MainWindow::close);
    connect(m_btnLogin, &QPushButton::clicked,
        this, &MainWindow::loginSuccesed);
}

void MainWindow::onValidate() {
    m_btnLogin->setEnabled(m_ileLogin->hasAcceptableInput() &&
        m_ilePassword->hasAcceptableInput());
}

void MainWindow::logged() {
    QMessageBox::information(nullptr, "Вход в систему",
        "Выполнен успешный вход в систему");
    close();
}

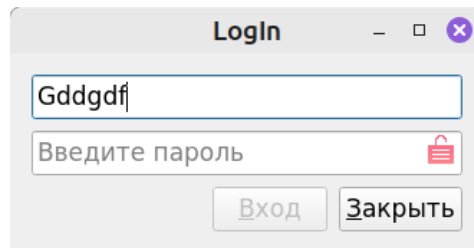
```

}

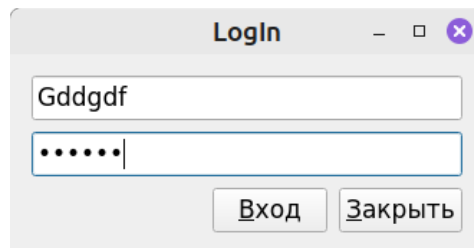
В методе `createUi()` полям ввода установлены валидаторы на основе регулярных выражений, а так же выполнено соединение сигналов об изменении текста полей ввода со слотом `onValidate()`.

Слот `onValidate()` делает доступной либо отключает кнопку Вход в зависимости от корректности введенных данных в поля ввода.

Окно приложения без введенного пароля (кнопка Вход отключена):



Окно приложения при введенном логине и пароле (кнопка Вход включена):



3. Программа отображения изображения.

В запущенной среде Qt Creator открываем меню Файл → Создать файл или проект... Выбираем Приложение (Qt) → Приложение Qt Widgets, нажимаем кнопку Choose... Указываем размещение проекта, задаем ему название Picture, нажимаем Далее, Далее. Класс виджета главного окна приложения называем MainWindow, выбираем базовый класс QWidget, форму окна приложения не генерируем. Pro-файл проекта, файл `main.cpp` и файлы класса MainWindow сгенерированы автоматически.

Копируем в каталог исходного кода проекта файлы `iconizedlineedit.h` и `iconizedlineedit.cpp` из проекта IconizedLineEdit и добавляем их как существующие файлы.

Заголовочный файл `mainwindow.h`:

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QWidget>

class IconizedLineEdit;
class QLabel;
class QVBoxLayout;

class MainWindow : public QWidget
{
```

```

        Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private slots:
    void getFile();

private:
    void loadPicture(const QString &fileName);

    IconizedLineEdit *m_ileFileName;
    QLabel *m_picture;
    QVBoxLayout *m_layout;
};
#endif // MAINWINDOW_H

```

В данный код внесено предварительное объявление классов QLabel, QVBoxLayout и IconizedLineEdit, объявление закрытого метода loadPicture(const QString &fileName) и слота getFile(), указателей на поля ввода имени файла класса IconizedLineEdit, надписи класса QLabel для отображения изображения и вертикального компоновщика.

Файл исходного кода mainwindow.cpp:

```

#include "mainwindow.h"
#include "iconizedlineedit.h"
#include <QVBoxLayout>
#include <QLabel>
#include <QFileDialog>

MainWindow::MainWindow(QWidget *parent)
    : QWidget(parent)
{
    m_layout = new QVBoxLayout;
    setLayout(m_layout);
    m_ileFileName = new IconizedLineEdit();
    m_ileFileName->setPlaceholderText("Нажмите, чтобы открыть файл");
    m_ileFileName->setIconVisiblity(IconizedLineEdit::IconAlwaysVisible);
    m_ileFileName->setIconPixmap(QPixmap("folder.png"));
    m_ileFileName->setIconClickable(true);
    connect(m_ileFileName, &IconizedLineEdit::iconPressed,
            this, &MainWindow::getFile);
    m_layout->addWidget(m_ileFileName);

    m_picture = new QLabel();
    m_layout->addWidget(m_picture);
}

MainWindow::~MainWindow() {
    delete m_layout;
    delete m_ileFileName;
    delete m_picture;
}

void MainWindow::getFile() {
    m_fileName = QFileDialog::getOpenFileName(this, "Открыть файл");
    m_ileFileName->setText(m_fileName);
    loadPicture(m_fileName);
}

void MainWindow::loadPicture(const QString &fileName) {

```

```

        m_picture->setPixmap(QPixmap(fileName).scaled(500, 500,
                                                    Qt::KeepAspectRatio));
    }

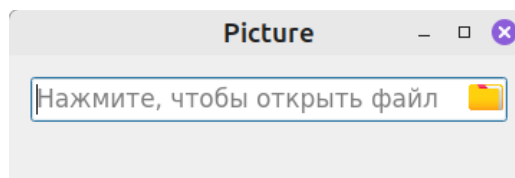
```

В коде конструктора создается поле ввода класса `IconizedLineEdit`, полю устанавливаются замещающий текст, тип видимости иконки (всегда видна), иконка (кликабельна), поле ввода добавляется в вертикальный компоновщик окна. Нажатие на иконке поля ввода соединяется со слотом `getFile()`. Создается надпись и добавляется в вертикальный компоновщик окна. Деструктор окна производит освобождение ресурсов компоновщика, поля ввода и надписи.

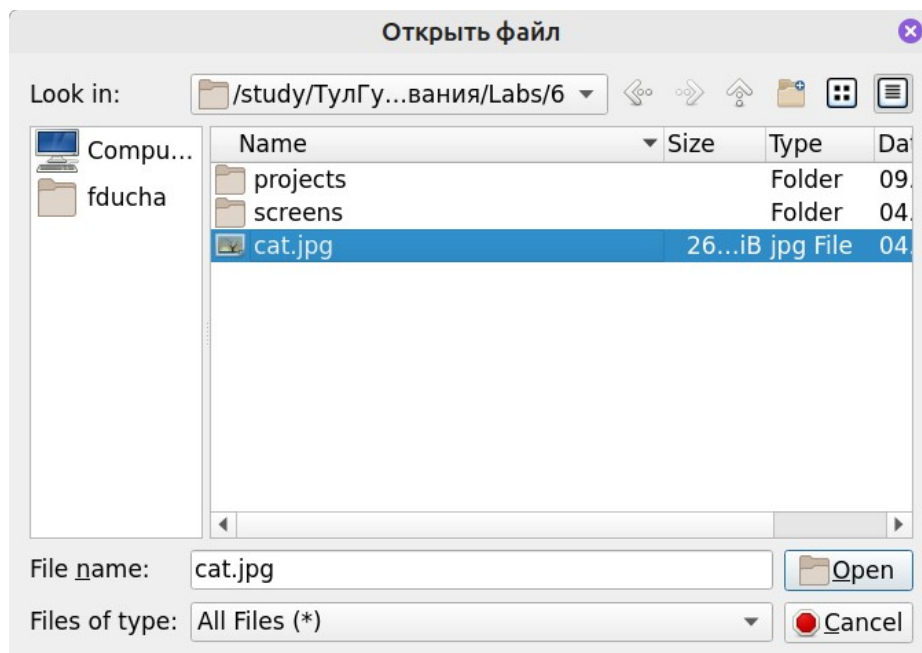
Слот `getFile()` открывает диалог открытия файла, получает имя файла изображения, устанавливает имя файла в поле ввода и вызывает метод `loadPicture` для отображения изображения.

Метод `loadPicture` получает имя файла, создает из него объект класса `QPixmap`, изменяет его размер до 500×500 пикселей с сохранением масштаба и устанавливает в надпись для отображения.

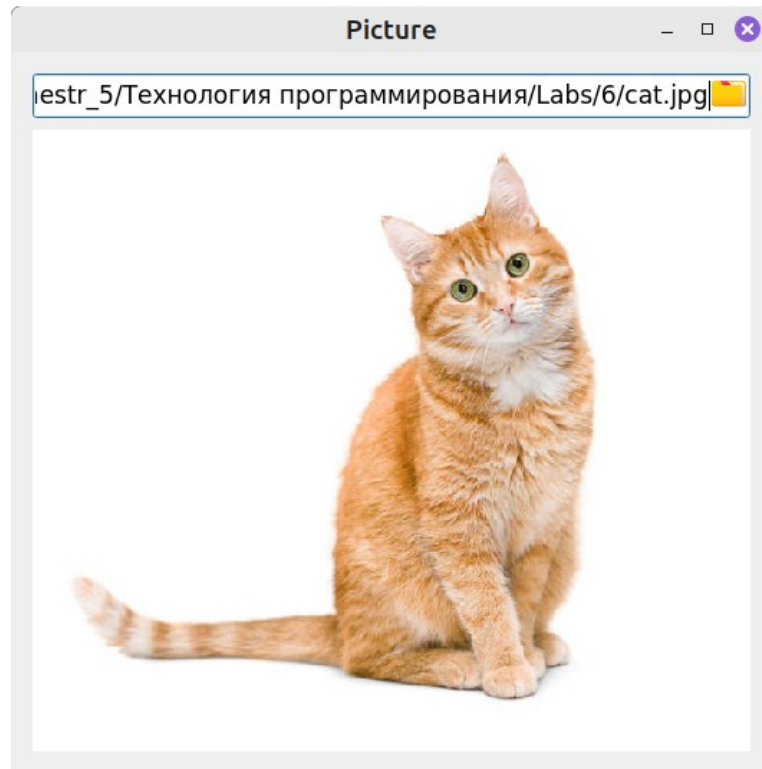
Окно приложения для выбора файла:



Диалог выбора файла:



Окно приложения с загруженным изображением:



4. Индикатор состояния.

В запущенной среде Qt Creator открываем меню Файл → Создать файл или проект... Выбираем Приложение (Qt) → Приложение Qt Widgets, нажимаем кнопку Choose... Указываем размещение проекта, задаем ему название LedTriIndicator, нажимаем Далее, Далее. Класс виджета главного окна приложения называем MainWindow, выбираем базовый класс QWidget, форму окна приложения не генерируем. Pro-файл проекта, файл main.cpp и файлы класса MainWindow сгенерированы автоматически.

Копируем в каталог исходного кода проекта файлы ledindicator.h и ledindicator.cpp из проекта LedIndicator и добавляем их как существующие файлы.

Заголовочный файл ledindicator.h:

```
#ifndef LEDINDICATOR_H
#define LEDINDICATOR_H

#include <QWidget>

class LedIndicator : public QWidget
{
    Q_OBJECT
    Q_PROPERTY(QString text READ text WRITE setText)
    Q_PROPERTY(bool turnedOn READ isTurnedOn WRITE setTurnedOn
        NOTIFY stateToggled)
    Q_PROPERTY(IndicatorState state READ state WRITE setState
        NOTIFY stateChanged)
```

```

public:
    enum IndicatorState {
        OFF = 0,
        MIDDLE,
        ON
    };

    explicit LedIndicator(QWidget *parent = nullptr);

    QString text() const;
    bool isTurnedOn() const;

    virtual QSize minimumSizeHint() const override;

    IndicatorState state() const;

signals:
    void stateToggled(bool);

    void stateChanged(IndicatorState state);

public slots:
    void setText(const QString &text);
    void setTurnedOn(bool turned);
    void setTristate(bool isTristate);

protected:
    virtual void paintEvent(QPaintEvent *event) override;

private slots:
    void setState(IndicatorState state);

private:
    QString m_text;
    bool m_isTristate{false};

    IndicatorState m_state{IndicatorState::OFF};
};

#endif // LEDINDICATOR_H

```

В код добавлено перечисление `IndicatorState` для описания 3 состояний индикатора, свойство `state` с методом получения состояния `state()`, закрытым слотом установки значения `setState(IndicatorState state)` и сигнал изменения состояния индикатора `stateChanged(IndicatorState state)`. Добавлен слот `setTristate(bool isTristate)` для установки режима работы либо с 2 состояниями, либо с 3. Режим работы с 2 или 3 состояниями хранится в переменной `m_isTristate`, а состояние индикатора в переменной `m_state` типа `IndicatorState`.

Файл исходного кода класса `LedIndicator`:

```

#include "ledindicator.h"
#include <QPainter>
#include <QPainterPath>
#include <QPaintEvent>

const int LED_RADIUS = 15;
const int LED_SPACING = 5;

LedIndicator::LedIndicator(QWidget *parent)
    : QWidget{parent}
{

```

```

}

QString LedIndicator::text() const {
    return m_text;
}

bool LedIndicator::isTurnedOn() const {
    return m_state == IndicatorState::ON || m_state == IndicatorState::MIDDLE;
}

QSize LedIndicator::minimumSizeHint() const {
    return QSize(LED_RADIUS * 2 + fontMetrics().horizontalAdvance(m_text) +
        LED_SPACING, LED_SPACING * 2);
}

void LedIndicator::setText(const QString &text) {
    m_text = text;
}

void LedIndicator::setTurnedOn(bool turned) {
    if (isTurnedOn() == turned) return;

    m_state = turned ? IndicatorState::ON : IndicatorState::OFF;
    emit stateToggled(turned);
    update();
}

void LedIndicator::setState(IndicatorState state) {
    if (m_state == state) return;

    m_state = state;
    emit stateChanged(m_state);
    update();
}

void LedIndicator::paintEvent(QPaintEvent *event) {
    Q_UNUSED(event);

    QPainter painter(this);
    painter.setRenderHint(QPainter::Antialiasing);

    QPoint ledCenter(LED_RADIUS + 1, height() / 2);

    QPainterPath path;
    path.addEllipse(ledCenter, LED_RADIUS, LED_RADIUS);
    painter.save();

    QRadialGradient gradient(ledCenter, LED_RADIUS);

    switch (m_state) {
    case IndicatorState::ON:
        painter.setPen(QPen(Qt::darkGreen));
        gradient.setColorAt(0.2, "#70FF70");
        gradient.setColorAt(1, "#00CC00");
        break;
    case IndicatorState::OFF:
        painter.setPen(QPen(Qt::black));
        gradient.setColorAt(0.2, Qt::gray);
        gradient.setColorAt(1, Qt::darkGray);
        break;
    case IndicatorState::MIDDLE:
        painter.setPen(QPen(Qt::red));
        gradient.setColorAt(0.2, Qt::yellow);
        gradient.setColorAt(1, Qt::red);
        break;
    }
}

```

```

    }

    painter.fillPath(path, QBrush(gradient));
    painter.drawPath(path);

    painter.restore();
    painter.setFont(font());

    QRect textRect(LED_RADIUS * 2 + LED_SPACING, 0,
                   width() - (LED_RADIUS * 2 + LED_SPACING), height());
    painter.drawText(textRect, Qt::AlignVCenter | Qt::AlignLeft, m_text);
}

void LedIndicator::setTristate(bool isTristate) {
    m_isTristate = isTristate;

    if (m_isTristate) setState(IndicatorState::MIDDLE);
    else setTurnedOn(false);
}

LedIndicator::IndicatorState LedIndicator::state() const {
    return m_state;
}

```

В данном коде сохранен интерфейс работы с 2 состояниями индикатора, то есть индикатор считается включенным, когда он включен или в промежуточном состоянии.

В перегруженном виртуальном методе `paintEvent` добавлена отрисовка промежуточного состояния индикатора (желтый цвет с красным краем). Данный метод работает с переменной состояния `m_state`.

При вызове метода `setTristate` с параметром `true`, индикатор сразу переходит в промежуточное состояние, а с параметром `false` — в выключенное.

Заголовочный файл главного окна `mainwindow.h`:

```

#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QWidget>
class QCheckBox;

class MainWindow : public QWidget
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

public slots:
    void unchecked();

private:
    QCheckBox *m_check;
    QCheckBox *m_checkMiddle;
};
#endif // MAINWINDOW_H

```


В код добавлен слот `unchecked()` и 2 указателя на виджеты класса `QCheckBox`: один включает / выключает индикатор, второй включает / выключает промежуточное состояние индикатора.

Файл исходного кода главного окна `mainwindow.cpp`:

```
#include "mainwindow.h"
#include "ledindicator.h"
#include <QBoxLayout>
#include <QCheckBox>

MainWindow::MainWindow(QWidget *parent)
    : QWidget(parent)
{
    auto layout = new QHBoxLayout;

    auto ledIndicator = new LedIndicator;
    ledIndicator->setMinimumHeight(50);
    ledIndicator->setText("Светофор");
    layout->addWidget(ledIndicator);

    m_check = new QCheckBox("LED ON");
    layout->addWidget(m_check);

    m_checkMiddle = new QCheckBox("SET MIDDLE");

    auto vlayout = new QVBoxLayout;
    setLayout(vlayout);
    vlayout->addLayout(layout);
    vlayout->addWidget(m_checkMiddle);

    connect(m_check, &QCheckBox::toggled, ledIndicator,
            &LedIndicator::setTurnedOn);
    connect(m_checkMiddle, &QCheckBox::toggled, ledIndicator,
            &LedIndicator::setTristate);
    connect(m_checkMiddle, &QCheckBox::toggled, this, &MainWindow::unchecked);
}

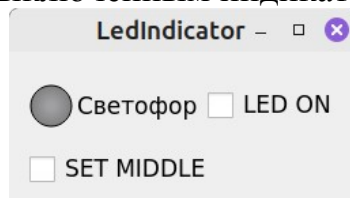
MainWindow::~MainWindow()
{
    delete m_check;
    delete m_checkMiddle;
}

void MainWindow::unchecked() {
    if (!m_checkMiddle->isChecked()) m_check->setChecked(false);
}
```

В конструкторе выполняется создание виджетов и их размещение в компоновщиках, а так же соединение опции включения индикатора со слотом `setTurnedOn`, опции включения промежуточного состояния со слотом `setTristate` и слотом `unchecked`, который отключает опцию включения индикатора при отключении опции включения промежуточного состояния.

Деструктор класса производит освобождение ресурсов указателей.

Окно приложения с выключенным индикатором:



Окно приложения с включенным индикатором:



Окно приложения с индикатором в промежуточном состоянии:



Промежуточное состояние так же включается при выключенном индикаторе:

