

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет информационной безопасности

Кафедра инфокоммуникационных технологий

**ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ
ИНФОКОММУНИКАЦИОННЫХ СИСТЕМ
Часть 1**

**Лабораторная работа 3
Массивы объектов. Работа с динамической памятью.
Слоты, сигналы и события**



Минск 2022

Содержание

Лабораторная работа 3	
Массивы объектов. Работа с динамической памятью. Слоты, сигналы и события.....	3
Одномерный динамический массив	3
Вложенные классы	5
Ключевое слово this.....	5
Массивы объектов	5
Слоты, сигналы и события.....	7
Задание к лабораторной работе 3.....	11

Лабораторная работа 3

Массивы объектов. Работа с динамической памятью.

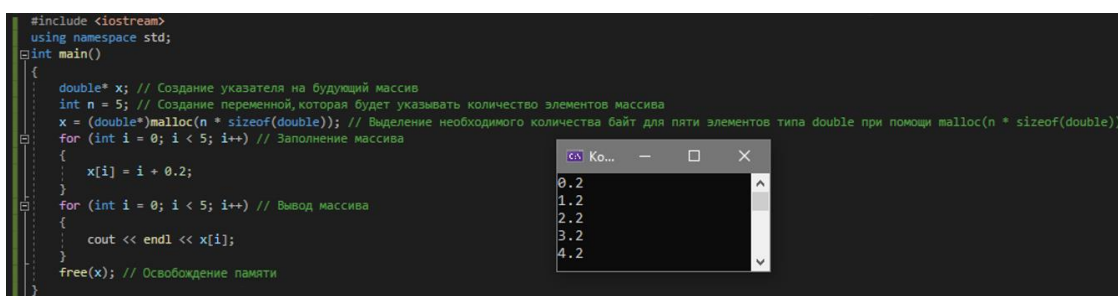
Слоты, сигналы и события

Цель работы: Изучить создание одномерных массивов при помощи конструктора с захватом динамической памяти и деструктора для их уничтожения. Научиться использовать механизм сигналов и слотов.

Одномерный динамический массив

При стандартной декларации массивов компилятор воспользуется векторной организацией памяти и выделит фиксированный участок памяти, достаточный для хранения всех его элементов. Использование динамической памяти и списковой ее организации позволяет создавать массивы переменной длины. Помимо уже известных операций по захвату и освобождению динамической памяти `new` и `delete` для этих целей используют стандартные библиотечные функции:

- `sizeof(type)` – возврат количества байт необходимого для хранения переменной типа, указанного в сигнатуре.
- `void* malloc(unsigned n)` – выделение памяти для размещения блока размером `n` байт; возвращает указатель на распределенную область или `NULL` при неудаче.
- `void free(void* b)` – освобождение блока памяти, адресуемого указателем `b`.

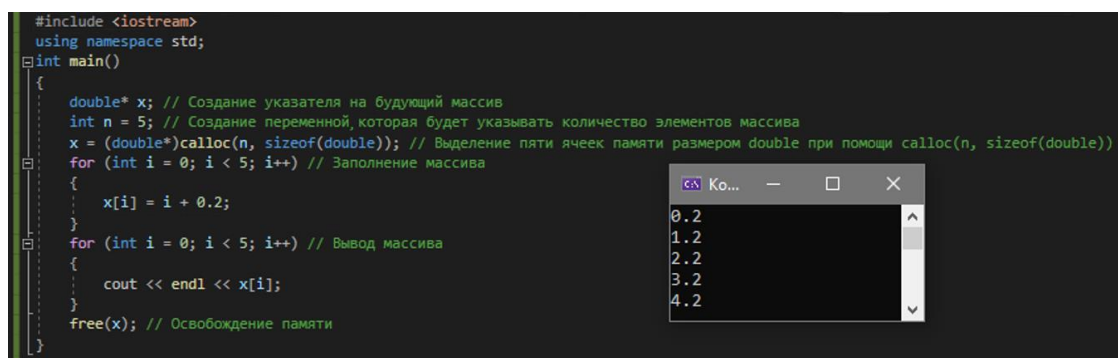


```
#include <iostream>
using namespace std;
int main()
{
    double* x; // Создание указателя на будущий массив
    int n = 5; // Создание переменной, которая будет указывать количество элементов массива
    x = (double*)malloc(n * sizeof(double)); // Выделение необходимого количества байт для пяти элементов типа double при помощи malloc(n * sizeof(double))
    for (int i = 0; i < 5; i++) // Заполнение массива
    {
        x[i] = i + 0.2;
    }
    for (int i = 0; i < 5; i++) // Вывод массива
    {
        cout << endl << x[i];
    }
    free(x); // Освобождение памяти
}
```

Консольный вывод:

```
0.2
1.2
2.2
3.2
4.2
```

- `void* calloc (unsigned n, unsigned size)` – выделение памяти для размещения `n` объектов размером `size` байт и заполнение полученной области нулями; возвращает указатель на захваченную область памяти или `NULL` при неудаче.



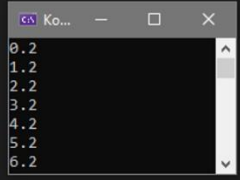
```
#include <iostream>
using namespace std;
int main()
{
    double* x; // Создание указателя на будущий массив
    int n = 5; // Создание переменной, которая будет указывать количество элементов массива
    x = (double*)calloc(n, sizeof(double)); // Выделение пяти ячеек памяти размером double при помощи calloc(n, sizeof(double))
    for (int i = 0; i < 5; i++) // Заполнение массива
    {
        x[i] = i + 0.2;
    }
    for (int i = 0; i < 5; i++) // Вывод массива
    {
        cout << endl << x[i];
    }
    free(x); // Освобождение памяти
}
```

Консольный вывод:

```
0.2
1.2
2.2
3.2
4.2
```

- `unsigned coreleft (void)` – получение размера свободной памяти (неиспользованной памяти) в байтах.
- `void* realloc(void* b, unsigne n)` – изменение размера блока, размещенного по адресу `b` на новое значение `n` с копированием (при необходимости) содержимого блока; возвращает указатель на перераспределенную область памяти или `NULL` при неудаче.

```
#include <iostream>
using namespace std;
int main()
{
    double* x; // Создание указателя на будущий массив
    int n = 5; // Создание переменной, которая будет указывать количество элементов массива
    x = (double*)malloc(n * sizeof(double)); // Выделение необходимого количества байт для пяти элементов типа double при помощи malloc(n* sizeof(double))
    for (int i = 0; i < 5; i++) // Заполнение массива
    {
        x[i] = i + 0.2;
    }
    x = (double*)realloc(x, (n + 2) * sizeof(double)); // Новое выделение блока памяти для x, с добавлением байт для хранения еще двух переменных.
    // При этом содержимое старого блока копируется в новый блок и информация не теряется */
    for (int i = 5; i < 7; i++) // Добавление новых значений элементам
    {
        x[i] = i + 0.2;
    }
    for (int i = 0; i < 7; i++) // Вывод массивов
        cout << endl << x[i];
    free(x); // Освобождение памяти
}
```



Общий формат операций `new` и `delete` для работы с динамической памятью имеет следующий вид.

```
type* name;
name = new type[size];
delete [] name;
```

где `type` – тип элементов, `size` – максимальное количество элементов массива `name`.

```
#include <iostream>
using namespace std;
int main()
{
    int* n; // Установка указателя на массив
    int lenght = 10; // Определение размера будущего массива
    n = new int[lenght]; // Выделение памяти указателю n, тип элементов массива - int, максимальное количество элементов массива - lenght
    delete[] n; // Освобождение памяти выделенной под массив
}
```

В программе ниже продемонстрирована реализация динамического массива в классе.

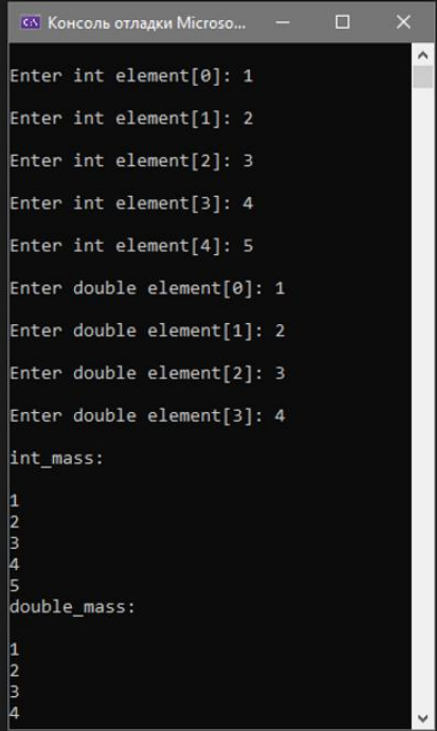
```
#include <iostream>
using namespace std;
class Test
{
public:
    // Указатели на массивы
    int* int_mass;
    double* double_mass;
    int int_lenght, double_lenght; // Переменные для хранения длины массивов
    Test(int r, int t); // Прототип конструктора с параметрами
    ~Test(); // Прототип деструктора
    void show();
};

Test::Test(int r, int t) // Реализация конструктора с параметрами
{
    int_mass = new int[r]; // Выделение памяти при помощи new
    int_lenght = r; // Определение размера массива типа int
    for (int i = 0; i < r; i++) // Заполнение массива типа int
    {
        cout << endl << "Enter int element[" << i << "]: ";
        cin >> int_mass[i];
    }
    double_mass = (double*)calloc(t, sizeof(double)); // Выделение памяти при помощи calloc
    double_lenght = t; // Определение размера массива типа double
    for (int i = 0; i < t; i++) // Заполнение массива типа double
    {
        cout << endl << "Enter double element[" << i << "]: ";
        cin >> double_mass[i];
    }
}

Test::~Test() // Реализация деструктора
{
    // Освобождение памяти
    delete[] int_mass;
    free(double_mass);
}

void Test::show()
{
    cout << endl << "int_mass:\n";
    // Вывод массива типа int
    for (int i = 0; i < int_lenght; i++)
    {
        cout << endl << int_mass[i];
    }
    cout << endl << "double_mass:\n";
    // Вывод массива типа double
    for (int i = 0; i < double_lenght; i++)
    {
        cout << endl << double_mass[i];
    }
}

int main()
{
    Test mass(5, 4); // Создание объекта mass класса Test. Передача размеров массивов в качестве параметров конструктору
    mass.show(); // Вывод массивов
}
```



Вложенные классы

Полями класса могут быть не только переменные базовых типов, но и объекты других классов.

```
#include <iostream>
using namespace std;
// Класс Discount будет использоваться в качестве поля класса Product
class Discount {
public:
    double pr_dis; // Поле процент скидки
    Discount() { // Конструктор без параметров
        pr_dis = 0;
        // Следующее сообщение выводится при создании объекта класса Discount
        cout << "Создание объекта Discount = " << pr_dis << endl;
    }
    ~Discount() { // Деструктор
        // Следующее сообщение выводится при удалении объекта класса Discount
        cout << "Удаление объекта Discount = " << pr_dis << endl;
    }
};

class Product { // Класс Product
public:
    double price; // Поле цена товара
    Discount percent; // Поле-объект типа Discount
    Product(double pr) { // Перегруженный конструктор с одним параметром
        price = pr; // Цена создаваемого товара не предусматривает скидки
        // Следующее сообщение выводится при создании объекта класса Product
        cout << "Создание товара с ценой (без учета скидки) = " << pr << endl;
    }
    Product(double pr, double dis) { // Перегруженный конструктор с двумя параметрами
        percent.pr_dis = dis;
        price = pr;
        cout << "Создание товара с ценой = " << price << endl;
        price = pr * (1 - percent.pr_dis); // Расчет цены товара с учетом скидки
    }
    ~Product() { //Деструктор
        // Следующее сообщение выводится при удалении объекта класса Product
        cout << "Удаление товара с ценой = " << price << endl;
    }
};

int main() {
    setlocale(LC_ALL, "Russian");
    Product product1(560); /* Создание объекта класса Product с помощью конструктора с одним параметром.
                           В качестве аргумента в этот конструктор передается значение 560 */
    cout << "Продукт 1: цена = " << product1.price << "Продукт 1: скидка = " << product1.percent.pr_dis << endl;
    // Вывод значений полей объектов
    cout << endl;
    Product product2(700, 0.3); /* Создание объекта класса Product с помощью конструктора с двумя параметрами.
                                В качестве аргументов в этот конструктор передаются значения 700 и 0.3 */
    cout << "Продукт 2: скидка = " << product2.percent.pr_dis << "Продукт 2: новая цена = " << product2.price << endl;
    // Вывод значений полей объектов
    cout << endl;
}
```

```
Консоль отладки Microsoft Visual Studio
Создание объекта Discount = 0
Создание товара с ценой (без учета скидки) = 560
Продукт 1: цена = 560 Продукт 1: скидка = 0

Создание объекта Discount = 0
Создание товара с ценой = 700
Продукт 2: скидка = 0.3 Продукт 2: новая цена = 490

Удаление товара с ценой = 490
Удаление объекта Discount = 0.3
Удаление товара с ценой = 560
Удаление объекта Discount = 0
```

Ключевое слово this

Существует один особый указатель, который неявно передается каждому методу класса. Это указатель на объект, из которого вызывается метод. Чтобы получить значение указателя на вызывающий метод объект, используют ключевое слово `this`.

Каждый раз, когда активизируется метод класса, он автоматически получает указатель с именем `this` на объект, для которого он вызван. Указатель `this` является неявным параметром всех методов. Таким образом, внутри методов `this` может быть использован для обращения к данному объекту.

```
#include <iostream>
using namespace std;
class Sale {
public:
    int price;
    int number;
    void show() {
        cout << "price = " << this->price << " ";
        cout << "number = " << this->number << endl;
    }
    void set(int p, int n) {
        this->price = p;
        this->number = n;
    }
};

int main() {
    Sale a, b;
    a.set(70, 20);
    b.set(80, 10);
    a.show();
    b.show();
}
```

```
Консоль отладки Microsoft Visual Studio
price = 69 number = 20
price = 88 number = 10
```

Массивы объектов

Массивы объектов можно создавать так же, как и массивы базовых типов данных. В качестве типа элементов массива указывается имя класса.

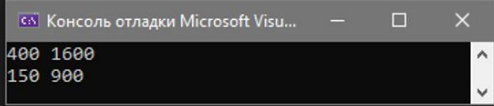
имя_класса имя_массива[размерность]

Обращение к объектам, являющимся элементами массива, осуществляется обычным образом с помощью индексов.

имя_массива[индекс].имя_элемента_класса

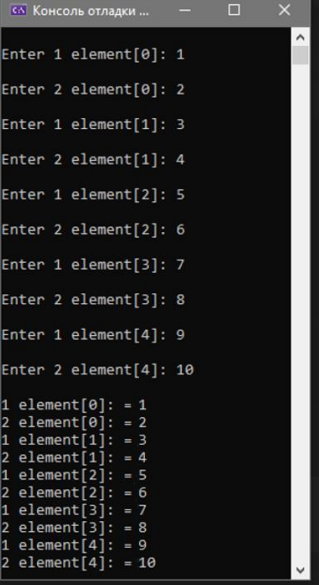
```
#include<iostream>
using namespace std;
class Sale {
public: // Открытые поля класса: цена товара и количество
    int price; int number;
    int total(); // Прототип метода
};
int Sale::total() // Реализация метода класса Sale
{
    return price * number;
}
int main() {
    Sale things[2]; // Создание массива объектов класса Sale
    things[0].price = 400; things[0].number = 4; // Присваивание значений полям 1-го объекта массива

    things[1].price = 150; things[1].number = 6; // Присваивание значений полям 2-го объекта массива
    // Вывод значений полей объектов
    cout << things[0].price << " ";
    cout << things[0].total() << endl;
    cout << things[1].price << " ";
    cout << things[1].total() << endl;
}
```



Массив объектов можно создавать разными способами. Например, используя объекты в качестве полей.

```
#include <iostream>
using namespace std;
class Mass_Element // Класс, объекты которого будут являться элементами массива
{
public:
    double a, b; // Поля для хранения значений типа double
};
class Test // Класс, объект которого будет хранить в себе массив объектов класса Mass_Element, его размер и методы для работы с этим массивом
{
public:
    int size; // Поле для хранения длины массива
    Mass_Element* mass; // Поле для хранения будущего массива объектов
    Test(int length); // Конструктор с параметрами
    ~Test(); // Деструктор
    void show(); // Метод для просмотра элементов массива
};
Test::Test(int length) // Реализация конструктора с параметрами
{
    mass = (Mass_Element*)calloc(length, sizeof(Mass_Element)); // Выделение памяти для массива объектов
    this->size = length; // Определение размера массива
    for (int i = 0; i < length; i++) // Заполнение элементов массива, которые в свою очередь являются объектами класса Test
    {
        cout << endl << "Enter 1 element[" << i << "]: "; cin >> mass[i].a; // Заполнение поля a объекта под номером i
        cout << endl << "Enter 2 element[" << i << "]: "; cin >> mass[i].b; // Заполнение поля b объекта под номером i
    }
}
Test::~Test()
{
    // Освобождение памяти
    free(mass);
}
void Test::show()
{
    for (int i = 0; i < this->size; i++) // Вывод поля каждого объекта массива
    {
        cout << endl << "1 element[" << i << "]: a = " << mass[i].a;
        cout << endl << "2 element[" << i << "]: b = " << mass[i].b;
    }
}
int main()
{
    Test mass(5); // Создание объекта, который хранит в себе массив объектов
    mass.show(); // Просмотр массива объектов
}
```



Программа ниже демонстрирует эту же реализацию, но в одном классе. В этом случае выделяется память элементу необходимая для хранения объекта класса Test, а в первом случае выделялась память для хранения объекта класса Mass_Element. Элемент должен хранить информацию только о a и b, поле для хранения size и указатель на массив, который прописан в классе Test.

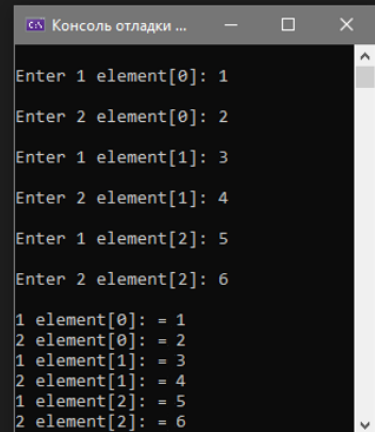
```
#include <iostream>
using namespace std;
class Test
{
public:
    double a, b; // Создание полей для хранения значений типа double
    int size; // Создание поля для хранения размера массива
    Test* mass; // Создание указателя для хранения будущего массива объектов
    Test(int lenght); // Создание прототипа конструктора с параметрами
    ~Test(); // Создание прототипа деструктора
    void show(); // Создание прототипа метода для демонстрации всех элементов массива
};

Test::Test(int lenght) // Реализация конструктора с параметрами
{
    mass = (Test*)calloc(lenght, sizeof(Test)); // Выделение памяти для массива объектов
    this->size = lenght; // Определение размера массива
    for (int i = 0; i < lenght; i++) // Заполнение элементов массива, которые в свою очередь являются объектами класса Test
    {
        cout << endl << "Enter 1 element[" << i << "]: "; cin >> mass[i].a; // Заполнение поля a объекта под номером i
        cout << endl << "Enter 2 element[" << i << "]: "; cin >> mass[i].b; // Заполнение поля b объекта под номером i
    }
}

Test::~Test()
{
    // Освобождение памяти
    free(mass);
}

void Test::show()
{
    for (int i = 0; i < this->size; i++) // Вывод поля каждого объекта массива
    {
        cout << endl << "1 element[" << i << "]: = " << mass[i].a;
        cout << endl << "2 element[" << i << "]: = " << mass[i].b;
    }
}

int main()
{
    Test mass(3); // Создание объекта, который хранит в себе массив объектов
    mass.show(); // Просмотр массива объектов
}
```



Слоты, сигналы и события

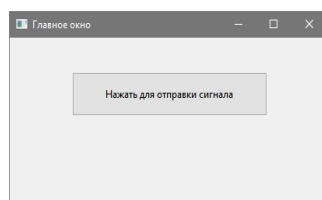
Механизм сигналов и слотов является расширением языка программирования C++ в Qt, который используется для установления связи между объектами. Если происходит какое-либо определенное событие, то при этом может генерироваться *сигнал*. Данный сигнал попадает в связанный с ним *слот*. В свою очередь, слот – это обычный метод в языке C++, который присоединяется к сигналу; он вызывается тогда, когда генерируется связанный с ним сигнал.

Взаимодействие окон

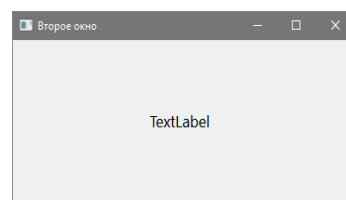
Создадим проект с несколькими формами, [смотреть](#). После создания форм реализуем взаимодействие сигналов и слотов.

Каждое окно – это объект определенного класса.

объект класса `MainWindow`



объект класса `Form`



Необходимо сделать так, чтобы при нажатии на кнопку в `Главном окне` изменялся текст во `Втором окне`. Для этого нужно при нажатии на кнопку отправить *сигнал*. Затем к *сигналу* подключить *слот*. Для того чтобы при появлении определенного сигнала вызывался определенный слот, сигнал и слот необходимо связать функцией `connect()`. Функция `connect(object1, signal1, object2, slot1,)` использует четыре аргумента. Два первых относятся к сигналу, два последних к слоту. Первый аргумент – это объект отправитель сигнала.

Второй аргумент – это сигнал. Третий аргумент – это объект в котором находится слот. Четвертый аргумент – это слот. Внесем изменения в файлы.

mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <form.h> // Соединим классы окон, подключаем заголовочный файл класса второго окна
QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

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

private slots:
    void on_pushButton_clicked();

private:
    Ui::MainWindow *ui;
    Form* form; // Создаем указатель на объект класса Form

signals:
    void signal(); // Создаем сигнал
};
#endif // MAINWINDOW_H
```

form.cpp

```
#include "form.h"
#include "ui_form.h"

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

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

void Form::slot()
{
    ui->label->setText("Сигнал получен"); // Меняем текст с помощью метода setText
}
```

form.h

```
#ifndef FORM_H
#define FORM_H

#include <QWidget>

namespace Ui {
class Form;
}

class Form : public QWidget
{
    Q_OBJECT

public:
    explicit Form(QWidget *parent = nullptr);
    ~Form();

private:
    Ui::Form *ui;

public slots:
    void slot(); // Создаем слот
};
#endif // FORM_H
```

mainwindow.cpp

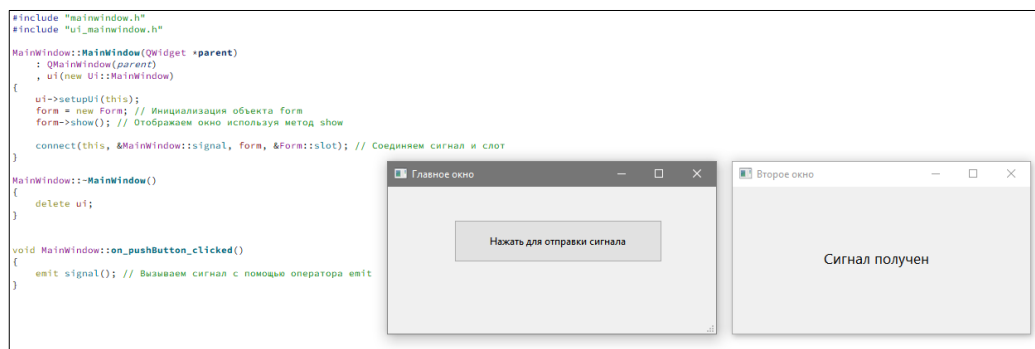
```
#include "mainwindow.h"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    form = new Form; // Инициализация объекта form
    form->show(); // Отображаем окно используя метод show
    connect(this, &MainWindow::signal, form, &Form::slot); // Соединим сигнал и слот
}

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

void MainWindow::on_pushButton_clicked()
{
    emit signal(); // Вызываем сигнал с помощью оператора emit
}
```

Теперь при нажатии на кнопку, текст во **Втором окне** изменяется.



Передадим в сигнале информацию, полученную от пользователя. Внесем изменения в форму **Главного окна**, [смотреть](#). Внесем изменения в файлы.

mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <form.h> // Соединим классы окон, подключаем заголовочный файл класса второго окна
QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

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

private slots:
    void on_pushButton_clicked();

private:
    Ui::MainWindow *ui;
    Form* form; // Создаем указатель на объект класса Form

signals:
    void signal(QString); // Создаем сигнал со строковым аргументом
};
#endif // MAINWINDOW_H
```

form.cpp

```
#include "form.h"
#include "ui_form.h"

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

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

void Form::slot(QString i)
{
    ui->label->setText(i); // Меняем текст с помощью метода setText. В качестве аргумента используем i
}
```

form.h

```
#ifndef FORM_H
#define FORM_H

#include <QWidget>

namespace Ui {
class Form;
}

class Form : public QWidget
{
    Q_OBJECT

public:
    explicit Form(QWidget *parent = nullptr);
    ~Form();

private:
    Ui::Form *ui;

public slots:
    void slot(QString i); // Создаем слот со строковым аргументом. Объект i класса QString
};
#endif // FORM_H
```

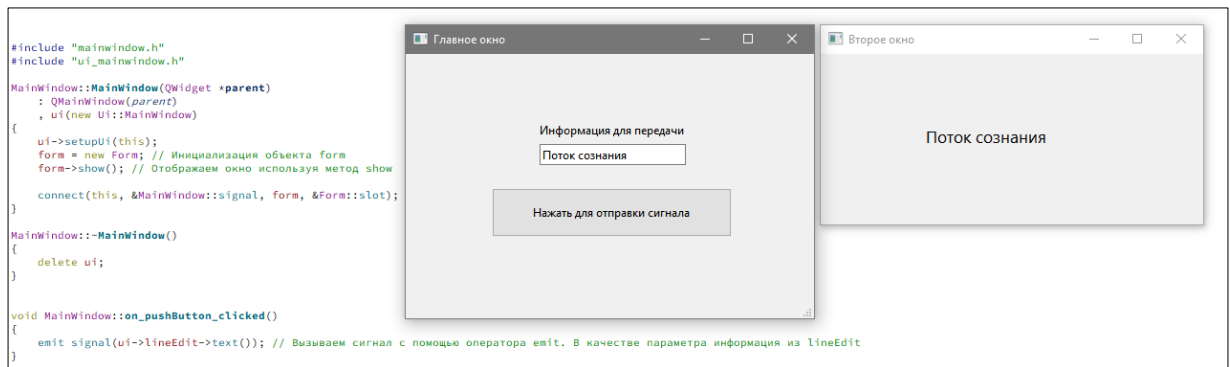
mainwindow.cpp

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

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    form = new Form; // Инициализация объекта form
    form->show(); // Отображаем окно используя метод show
    connect(this, &MainWindow::signal, form, &Form::slot); // Соединим сигнал и слот
}

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

void MainWindow::on_pushButton_clicked()
{
    emit signal(ui->lineEdit->text()); // Вызываем сигнал с помощью оператора emit. В качестве параметра информации из lineEdit
}
```

Передадим информацию из Второго окна на Главное окно. Внесем изменения в формы, [смотреть](#) и файлы.

mainwindow.h

```

#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <form.h> // Соединяем классы окон, подключаем заголовочный файл класса второго окна

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

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

private slots:
    void on_pushButton_clicked();

private:
    Ui::MainWindow *ui;
    Form *form; // Создаем указатель на объект класса Form

signals:
    void signal(QString); // Создаем сигнал со строковым аргументом

public slots:
    void slotForm(QString i); // Создаем слот со строковым аргументом. Объект i класса QString
};

#endif // MAINWINDOW_H

```

form.cpp

```

#include "form.h"
#include "ui_form.h"

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

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

void Form::slot(QString i)
{
    ui->label->setText(i); // Меняем текст с помощью метода setText. В качестве аргумента используем i
}

void Form::on_pushButton_clicked()
{
    emit signalForm(ui->lineEdit->text()); // Вызываем сигнал с помощью оператора emit. В качестве параметра информация из QLineEdit
}

```

form.h

```

#ifndef FORM_H
#define FORM_H

#include <QWidget>

namespace Ui {
class Form;
}

class Form : public QWidget
{
    Q_OBJECT

public:
    explicit Form(QWidget *parent = nullptr);
    ~Form();

private:
    Ui::Form *ui;

public slots:
    void slot(QString i); // Создаем слот со строковым аргументом. Объект i класса QString

signals:
    void signalForm(QString); // Создаем сигнал со строковым аргументом.

private slots:
    void on_pushButton_clicked();
};

#endif // FORM_H

```

mainwindow.cpp

```

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

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    form = new Form; // Инициализация объекта form
    form->show(); // Отображаем окно используя метод show

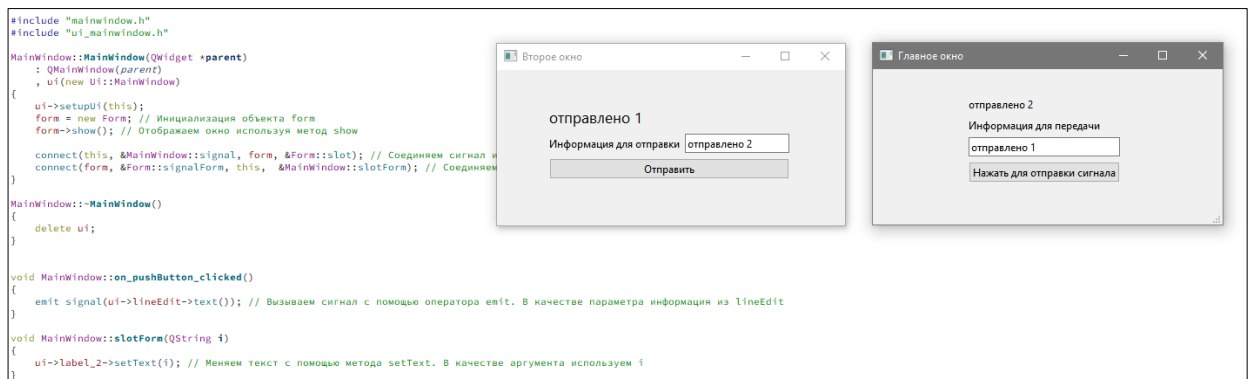
    connect(this, &MainWindow::signal, form, &Form::slot); // Соединяем сигнал и слот
    connect(form, &Form::signalForm, this, &MainWindow::slotForm); // Соединяем сигнал и слот
}

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

void MainWindow::on_pushButton_clicked()
{
    emit signal(ui->lineEdit->text()); // Вызываем сигнал с помощью оператора emit. В качестве параметра информация из QLineEdit
}

void MainWindow::slotForm(QString i)
{
    ui->label_2->setText(i); // Меняем текст с помощью метода setText. В качестве аргумента используем i
}

```



Сигналы являются публично доступными функциями и могут быть вызваны где угодно, но рекомендуется их вызывать только в классе, где они были определены, а также в его подклассах.

Когда сигнал вызван, слот подключенный к нему обычно выполняется незамедлительно, просто как нормальная функция. Это возможно потому, что механизм сигналов и слотов является независимым от каких-либо циклов в GUI. Выполнение кода следует вызывать оператором **emit**.

Взаимодействие компонентов

Создадим окно с кнопками `Increase`, `Decrease` и объектом `QLabel` (значение 0), скомпонованными по сетке, [смотреть](#). После создания компонентов реализуем взаимодействие сигналов и слотов.

mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>

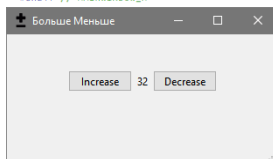
QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

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

private:
    Ui::MainWindow *ui;
private slots:
    void on_pushButton_1_clicked();
    void on_pushButton_2_clicked();
};

#endif // MAINWINDOW_H
```



mainwindow.cpp

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

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    // Соединим сигналы с соответствующими слотами
    connect(ui->pushButton_1, SIGNAL(clicked()), this, SLOT(on_pushButton_1_clicked()));
    connect(ui->pushButton_2, SIGNAL(clicked()), this, SLOT(on_pushButton_2_clicked()));
}

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

/* В методе определяем текущее значение в result.
 * result отображает строковое значение, поэтому сначала его нужно преобразовать в целое число.
 * Затем это число увеличиваем, конвертируем получившееся число в строковое значение и устанавливаем
 * новый текст для result */
void MainWindow::on_pushButton_1_clicked()
{
    int val=ui->result->text().toInt(); // Выполняем конвертацию в int
    val++; // Выполняем инкремент значения
    ui->result->setText(QString::number(val)); // Конвертируем обратное в QString и устанавливаем новый текст для result
}

/* В методе определяем текущее значение в result.
 * result отображает строковое значение, поэтому сначала его нужно преобразовать в целое число.
 * Затем это число увеличиваем, конвертируем получившееся число в строковое значение и устанавливаем
 * новый текст для result */
void MainWindow::on_pushButton_2_clicked()
{
    int val=ui->result->text().toInt(); // Выполняем конвертацию в int
    val--; // Выполняем декремент значения
    ui->result->setText(QString::number(val)); // Конвертируем обратное в QString и устанавливаем новый текст для result
}
```

Задание к лабораторной работе 3

Для выполнения лабораторной работы необходимо установить и настроить Visual Studio и инструменты Qt. Задание к лабораторной работе 3 состоит из нескольких задач. Задачи выполняются по вариантам (например, Вариант 1 – номера по списку в группе: 1, 9, 17, 26; Вариант 2 – номера по списку в группе: 2, 10, 18, 27 и т.д.).

Вариант 1



Задача 1

Задача 1 выполняется с использованием Visual Studio. Пользовательский класс должен содержать конструктор с параметром для создания динамических массивов (оператор `new` или стандартная библиотечная функция `calloc`) и установки начальных значений элементов: (размер массива передается как аргумент для конструктора), так же класс должен иметь метод для просмотра текущего состояния массива и деструктор, который освободит память.

На вашу почту пришел заказ от компании "Cool developer". Вас просят разработать инвентарь персонажа для хранения предметов. Так как игра в стадии разработки Pre-Alpha – Alpha – Beta – Release Candidate, то названия у предметов нет, есть только ID предметов. Вам необходимо создать в классе `Inventory` массив размером `size` (`size` – количество ячеек в инвентаре) и заполнить его случайными значениями от 0 до 100000. Реализовать метод `run`, который выведет количество элементов с нечетным ID.

Как только заказчик увидел, что все ячейки заполнены какими-то непонятными ему числами, он дал команду придумать название каждому предмету и задать вес для каждого предмета. Теперь вам нужно все переделывать с учетом новых данных. Вам необходимо в созданном ранее классе создать массив объектов этого класса и добавить поля: `ID`, `weight` и `name`. `weight` – генерируется случайно в диапазоне от 1 до 100. Метод `run` должен вывести всю информацию о каждом объекте, лежащем в инвентаре, у которого $30 < \text{weight} < 90$. Поля `ID` и `name` вводятся вручную. Размер массива передается в конструктор в качестве параметра, заполнение массива и информации о каждом объекте происходит в том же конструкторе.

Задача 2

Задача 2 выполняется с использованием инструментов Qt. Необходимо дать поясняющие комментарии к коду.

Заказчик попросил вас создать приложение для расчета суммы предметов в инвентаре. Количество предметов необходимо вводить в два поля (Базовый рюкзак, Сумка), расположенные в главном окне. Вывод суммы предметов осуществлять в другое модальное окно.

Вариант 2



Задача 1

Задача 1 выполняется с использованием Visual Studio. Пользовательский класс должен содержать конструктор с параметром для создания динамических массивов (оператор `new` или стандартная библиотечная функция `calloc`) и установки начальных значений элементов: (размер массива передается как аргумент для конструктора), так же класс должен иметь метод для просмотра текущего состояния массива и деструктор, который освободит память.

После того, как вы доказали свой профессионализм начальству, вас перевели в лабораторию астрофизики. Первым заданием стало записать расстояния от Солнца до звезд, попадающих в область видимости телескопа "James Webb". Пачку бумаги вам снова не захотелось, и вы приступили к работе. Вам необходимо создать класс `Stars`, а в нем массив размером `size` (`size` – количество звезд которое наблюдает "James Webb") и заполнить его случайными значениями от 0 до 999999. Реализовать метод `run`, который выведет разность между самой дальней и самой близкой звездой (разумеется в космосе для оценки межзвездных расстояний используются парсеки).

Одного расстояния до других звезд недостаточно, необходимо добавить еще больше данных. Вам необходимо в созданном ранее классе создать массив объектов этого класса и добавить поля: `distance`, `name` и `size`. `distance` и `size` – генерируется случайно в диапазоне от 1 до 100000. Метод `run` должен вывести всю информацию о каждом объекте, лежащем в диапазоне $400 < \text{distance} < 90000$. Поле `name` вводится вручную. Размер массива передается в конструктор в качестве параметра, заполнение массива и информации о каждом объекте происходит в том же конструкторе.

Задача 2

Задача 2 выполняется с использованием инструментов Qt. Необходимо дать поясняющие комментарии к коду.

Коллеги из лаборатории астрофизики попросили вас написать приложение для расчета разности между самой дальней и самой близкой звездой. Расстояние звезд необходимо вводить в два поля (самая дальняя звезда, самая близкая звезда), расположенные в главном окне. Вывод разности расстояния звезд осуществлять в другое модальное окно.

Вариант 3



Задача 1

Задача 1 выполняется с использованием Visual Studio. Пользовательский класс должен содержать конструктор с параметром для создания динамических массивов (оператор `new` или стандартная библиотечная функция `calloc`) и установки начальных значений элементов: (размер массива передается как аргумент для конструктора), так же класс должен иметь метод для просмотра текущего состояния массива и деструктор, который освободит память.

Завтра в вашу больницу приезжает проверка, которая будет смотреть на заполнение палат в больнице. Чтобы не ходить по палатам и считать количество пациентов вручную, вы решили написать программу, в которой будет храниться информация о количестве человек в каждой палате. Вам необходимо создать класс `Patients`, в котором будет массив размером `size` (`size` – количество палат в больнице) и заполнить его случайными значениями от 0 до 10. Реализовать метод `run`, который выведет номера палат, в которых количество пациентов >5 (так как эта современная больница, то номера палат начинаются с 0, а то программисты, которым говорили идти в самую первую палату, бродили по больнице и не могли найти палату с номером 0).

Проверка прошла успешно. Идея с программой, в которой хранится информация была хороша, но хотелось бы получить информацию и о каждом пациенте в палате. Вам необходимо в разработанном ранее классе создать массив объектов этого класса и добавить поля: `duration_of_observation`, `name` и `ward_number`. `duration_of_observation` – генерируется случайно в диапазоне от 1 до 90. Метод `run` должен вывести всю информацию о каждом объекте, у которого $20 < \text{duration_of_observation} < 70$. Поля `name` и `ward_number` вводятся вручную. Размер массива передается в конструктор в качестве параметра, заполнение массива и информации о каждом объекте происходит в том же конструкторе.

Задача 2

Задача 2 выполняется с использованием инструментов Qt. Необходимо дать поясняющие комментарии к коду.

Для лучшего восприятия информации о количестве пациентов, вам необходимо написать приложение для их подсчета. Количество пациентов в палатах необходимо вводить в пять полей (палата №501, палата №502, палата №503, палата №504, палата №505), расположенные в главном окне. Вывод информации о количестве всех пациентов осуществлять в другое модальное окно.

Вариант 4



Задача 1

Задача 1 выполняется с использованием Visual Studio. Пользовательский класс должен содержать конструктор с параметром для создания динамических массивов (оператор `new` или стандартная библиотечная функция `calloc`) и установки начальных значений элементов: (размер массива передается как аргумент для конструктора), так же класс должен иметь метод для просмотра текущего состояния массива и деструктор, который освободит память.

Ваш начальник хочет знать, сколько каждый клиент, вошедший в его банк, потратил денег. Желание начальника – закон. Вы приступаете к работе. Вам необходимо создать класс `Customers`, а в нем массив размером `size` (`size` – количество клиентов, посетивших банк за день) и количеством денег которое потратил каждый клиент, значение которого генерируется рандомно в диапазоне от 0 до 100000. Реализовать метод `run`, который выведет общее количество денег, полученное банком от всех клиентов, посетивших его за день. Так же выведет номера клиентов, которые потратили больше 90000 (начальник лично хочет выдать им карточки VIP-клиентов, в благодарность за свою новую Tesla).

Вашему начальнику зачем-то необходимо знать еще возраст и пол клиентов банка. Ну раз надо, значит будет. Приступаем. Вам необходимо в разработанном ранее классе создать массив объектов этого класса и добавить поля: `spending`, `age` и `gender` (типа `bool`, мужчины `==0`, девушки `==1`). `age` – генерируется рандомно в диапазоне от 16 до 90. `spending` – генерируется рандомно в диапазоне от 0 до 100000. Метод `run` должен вывести всю информацию о каждом объекте, у которого $18 < \text{age} < 30$ и $25000 < \text{spending} < 100000$ и `gender == 1`. Поле `gender` вводится вручную. Размер массива передается в конструктор в качестве параметра, заполнение массива и информации о каждом объекте происходит в том же конструкторе.

Задача 2

Задача 2 выполняется с использованием инструментов Qt. Необходимо дать поясняющие комментарии к коду.

Ваш начальник пожелал увидеть количество денег, полученное банком от клиентов за неделю. Вам необходимо написать приложение для их подсчета. Количество денег, полученное от клиентов в день необходимо вводить в поля (Понедельник, Вторник, Среда, Четверг, Пятница, Суббота, Воскресенье), расположенные в главном окне. Вывод информации о количестве всех денег за неделю осуществлять в другое модальное окно.

Вариант 5



Задача 1

Задача 1 выполняется с использованием Visual Studio. Пользовательский класс должен содержать конструктор с параметром для создания динамических массивов (оператор `new` или стандартная библиотечная функция `calloc`) и установки начальных значений элементов: (размер массива передается как аргумент для конструктора), так же класс должен иметь метод для просмотра текущего состояния массива и деструктор, который освободит память.

Намечаются чудесные выходные, и вы бы не хотели, чтобы их что-нибудь испортило, к примеру, бунт в камерах. Бунт в камере может начаться, если в ней сидит >8 человек. Для того, чтобы узнать сколько сидит заключенных в камере, Вы решили написать программу. Вам необходимо создать класс `Criminals`, а в нем массив размером `size` (`size` – количество камер) и заполнить его случайными значениями от 0 до 10. Реализовать метод `run`, который выведет номера камер с количеством заключенных >8 (так как недавно был день программиста и все камеры забиты исключительно представителями этой профессии, то вам пришлось переделать нумерацию камер начиная с 0, иначе они отказывались задерживаться в камерах на положенные 15 суток).

Вы уже собирались домой, но вдруг услышали возмущенные крики программистов "ООП – рулит. Мы все, объекты. HTML – не язык программирования". Ну что ж, с последним вы согласны, а что делать с их заявлением "Мы все объекты"? "Придется сделать их всех объектами", подумали вы и приступили к работе. Вам необходимо в разработанном ранее классе создать массив объектов этого класса и добавить поля: `experience`, `name` и `level_in_the_skyrim`. `experience` – генерируется случайно в диапазоне от 0 до 100. `level_in_the_skyrim` – генерируется случайно в диапазоне от 0 до 255. Метод `run` должен вывести всю информацию о каждом объекте, у которого `level_in_the_skyrim`>50 и `experience`>30 (может вы их и отпустите). Поле `name` вводится вручную. Размер массива передается в конструктор в качестве параметра, заполнение массива и информации о каждом объекте происходит в том же конструкторе.

Задача 2

Задача 2 выполняется с использованием инструментов Qt. Необходимо дать поясняющие комментарии к коду.

Для лучшего восприятия информации о количестве заключенных, вам необходимо написать приложение для их подсчета. Количество заключенных в камерах необходимо вводить в четыре поля (камера 11, камера 12, камера 13, камера 14), расположенные в главном окне. Вывод информации о количестве всех заключенных осуществлять в другое модальное окно.

Вариант 6



Задача 1

Задача 1 выполняется с использованием Visual Studio. Пользовательский класс должен содержать конструктор с параметром для создания динамических массивов (оператор `new` или стандартная библиотечная функция `calloc`) и установки начальных значений элементов: (размер массива передается как аргумент для конструктора), так же класс должен иметь метод для просмотра текущего состояния массива и деструктор, который освободит память.

Посмотрев в казну своего города, вы обнаружили, что пауки при помощи паутины сплели на потолке слово "Пустота". Подумав немного, вы решили узнать, куда делись все деньги бюджета путем составления списка с зарплатами всех работников мэрии. Вам необходимо создать класс `Workers`, а в нем массив размером `size` (`size` – количество сотрудников мэрии) и зарплата каждого сотрудника, которая генерируется случайно в диапазоне от 100 до 999999. Реализовать метод `run`, который урежет зарплату всем, у кого она >1000 , в 2 раза и выведет эти новые зарплаты сотрудников.

Вы вроде бы урезали зарплату сотрудникам, а денег как не было, так и нет.

Подумав немного, вы все же поняли "Сотрудники то, подворовывают". Теперь необходимо узнать кто и сколько. Вам необходимо в разработанном ранее классе создать массив объектов этого класса и добавить поля: `stolen`, `name` и `gender` (типа `bool`, мужчины $==0$, девушки $==1$). `stolen` – генерируется случайно в диапазоне от 0 до 100000. Метод `run` должен вывести всю информацию о каждом объекте, у которого `stolen > 10000` и `gender == 0` или `stolen > 20000` и `gender == 1` (их придется уволить). Поле `name` и `gender` вводится вручную. Размер массива передается в конструктор в качестве параметра, заполнение массива и информации о каждом объекте происходит в том же конструкторе.

Задача 2

Задача 2 выполняется с использованием инструментов Qt. Необходимо дать поясняющие комментарии к коду.

Для лучшего восприятия информации о количестве сэкономленных денег на зарплату, вам необходимо написать приложение для их подсчета. Количество денег необходимо вводить в два поля (сумма, поступившая в бюджет, сумма, потраченная на зарплату), расположенные в главном окне. Вывод информации о количестве сэкономленных денег осуществлять в другое модальное окно.

Вариант 7



Задача 1

Задача 1 выполняется с использованием Visual Studio. Пользовательский класс должен содержать конструктор с параметром для создания динамических массивов (оператор `new` или стандартная библиотечная функция `calloc`) и установки начальных значений элементов: (размер массива передается как аргумент для конструктора), так же класс должен иметь метод для просмотра текущего состояния массива и деструктор, который освободит память.

23-е февраля закончилось, и теперь ваша фабрика носков потеряла все свои заказы. Но вы не унываете, ибо скоро 8-е марта, а фабрика теперь может выпускать полосатые гольфы тех цветов, которые выберет заказчик. Но все хотят разное количество полосок, и чтобы не запутаться, нужна программа. Вам необходимо создать класс `Half_hose`, а в нем массив размером `size` (`size` – количество гольфов) и количество полосок в каждом гольфе которое генерируется случайно в диапазоне от 3 до 1001. Реализовать метод `run`, который добавит 1 полоску в каждый гольф, в котором будет четное число полосок и вывести новые значения полосок в каждом гольфе.

Вы уже почти начали производство полосатых гольфов, но вдруг вспомнили, что не знаете кому какие доставлять. Вам необходимо срочно записать адрес доставки каждой пары гольф, определить цену и количество полос, чтобы в будущем быстро узнать свою прибыль. Ваша задача: в разработанном ранее классе создать массив объектов этого класса и добавить поля: `address`, `price`, `qty`. `qty` – генерируется случайно в диапазоне от 1 до 1001. `price` и `address` – вводится вручную. Размер массива передается в конструктор в качестве параметра, заполнение массива и информации о каждом объекте происходит в том же конструкторе.

Задача 2

Задача 2 выполняется с использованием инструментов Qt. Необходимо дать поясняющие комментарии к коду.

Для лучшего восприятия информации о количестве всех проданных гольфов, вам необходимо написать приложение для их подсчета. Количество гольфов необходимо вводить в четыре поля (гольфы с двумя полосками, гольфы с тремя полосками, гольфы с четырьмя полосками, гольфы с пятью полосками), расположенные в главном окне. Вывод информации о количестве всех проданных гольфов осуществлять в другое модальное окно.

Вариант 8



Задача 1

Задача 1 выполняется с использованием Visual Studio. Пользовательский класс должен содержать конструктор с параметром для создания динамических массивов (оператор `new` или стандартная библиотечная функция `calloc`) и установки начальных значений элементов: (размер массива передается как аргумент для конструктора), так же класс должен иметь метод для просмотра текущего состояния массива и деструктор, который освободит память.

Для привлечения всех категорий граждан в отель Вы придумали новую опцию "Своя комната". Клиент говорит, сколько он готов отдать за комнату и исходя от предложенной им суммы, ему становятся доступны некие услуги отеля и комплекты мебели для обстановки комнаты. Теперь необходимо посчитать прибыль от такого нововведения. Вам необходимо создать класс `Hostel`, а в нем массив размером `size` (`size` – количество сданных комнат в отеле) и заполненный ценой каждой комнаты, которая генерируется случайно в диапазоне от 100 до 8000. Реализовать метод `run`, который выведет прибыль от всех комнат.

Ваша идея увенчалась успехом, отбоя от желающих просто нет, но есть проблема. Вы обнаружили, что один человек заказывает для себя абсолютно пустую комнату и платит за нее копейки, а живут в этой комнате 12 человек. Хитро, но вы хитрее. Необходимо проверить все комнаты и выселить неплательщиков. Вам необходимо в разработанном ранее классе создать массив объектов этого класса и добавить поля: `number_of_residents`, `price` и `number_of_beds`. `price` – генерируется случайно в диапазоне от 100 до 100000. Метод `run` должен вывести всю информацию о каждом объекте, у которого `number_of_residents > number_of_beds` (их точно нужно выселить). Поля `number_of_residents` и `number_of_beds` вводятся вручную. Размер массива передается в конструктор в качестве параметра, заполнение массива и информации о каждом объекте происходит в том же конструкторе.

Задача 2

Задача 2 выполняется с использованием инструментов Qt. Необходимо дать поясняющие комментарии к коду.

Для лучшего восприятия информации о количестве прибыли от сдачи комнат, вам необходимо написать приложение для ее подсчета. Количество прибыли необходимо вводить в три поля (своя комната №101, своя комната №201, своя комната №301), расположенные в главном окне. Вывод информации о количестве прибыли осуществлять в другое модальное окно.