Лабораторная работа №2
по курсу «Высокоуровневое программирование» (2 семестр)
«Наследование и иерархия классов»

Оглавление

Цель: приобретение практических навыков проектирования иерархии классов.

Задачи:

1. Изучить понятие иерархии классов.
2. Познакомиться со способами наследования классов.
3. Изучить понятие абстрактный класс.
4. Изучить понятие интерфейса.
5. Изучить работу с виртуальными функциями.
6. Научиться выполнять инициализацию объекта разными способами.

Содержание отчёта:

1. Титульный лист;
2. Цель, задачи работы;
3. Формулировка общего задания;
4. UML-диаграмма классов.
5. Листинги пользовательских функций, классов и основной программы;
6. Результаты работы программы.
7. Выводы по работе в целом.

В начало

Основные теоретические сведения

Наследование

Наследование – это одно из средств разработки программ, используемых ООП технологию, которое позволяет не повторять код много раз, а также построить реальную иерархию сущностей для удобного манипулирования ими. На прошлой лабораторной работе уже было рассмотрено простое создание класса, его полей и методов. Также было затронуто такое понятие, как модификаторы доступа, реализующие принцип инкапсуляции. Концепцию наследования мы рассмотрим на реальном пример. Попробуем написать класс банковского работника. Для большей наглядности класс мы будем писать в файле исходного кода без отделения прототипа от реализации (но только для наглядности!):

```
class BankWorker {

public:
	typedef char* string;
	typedef unsigned int score;

	enum class Gender {
		male,
		female
```

```
        };

        string getName() {
                return m_name;
        }
        score getAge() {
                return m_age;
        }
        Gender getGender() {
                return m_gender;
        }
        string getPosition() {
                return m_position;
        }
        score getSalary() {
                return m_salary;
        }

        void setName() {
                /* реализация */
        }
        void setAge() {
                /* реализация */
        }
        void setGender() {
                /* реализация */
        }
        void setPosition() {
                /* реализация */
        }
        void setSalary() {
                /* реализация */
        }

private:
        string m_name{};
        score m_age{};
        Gender m_gender{};
        string m_position{};
        score m_salary{};
};
```

Давайте рассмотрим поля класса `BankWorker`. Мы видим, что некоторые из них относятся ко всем людям: возраст, пол, имя. И если нам придётся писать ещё одну структуру, например студента, то мы будем обречены прописывать ещё раз все эти поля. В таком случае нам просто напрашивается мысль – создать ещё одну сущность – `Human`, которая будет инкапсулировать (содержать и прятать в себе все свойства характерные для человека).

```
class Human {

public:
        typedef char* string;
        typedef unsigned int score;

        enum class Gender {
                male,
                female
```

```
        };

        string getName() {
                return m_name;
        }
        score getAge() {
                return m_age;
        }
        Gender getGender() {
                return m_gender;
        }

        void setName() {
                /* реализация */
        }
        void setAge() {
                /* реализация */
        }
        void setGender() {
                /* реализация */
        }

private:
        string m_name{};
        score m_age{};
        Gender m_gender{};
};
```

Мы создали класс – `Human`, который имеет поля: имя, возраст и пол. Теперь первый класс мы можем переписать так:

```
class BankWorker {

public:

        string getPosition() {
                return m_position;
        }
        score getSalary() {
                return m_salary;
        }

        void setPosition() {
                /* реализация */
        }
        void setSalary() {
                /* реализация */
        }

private:
        string m_position{};
        score m_salary{};
};
```

Посмотрите, на сколько уменьшился код нашего класса. И теперь самое главное, как соединить эти два класса вместе? В таких случаях применяется *наследование*.

```
class BankWorker : public Human {

public:

	string getPosition() {
		return m_position;
	}
	score getSalary() {
		return m_salary;
	}

	void setPosition() {
		/* реализация */
	}
	void setSalary() {
		/* реализация */
	}

private:
	string m_position{};
	score m_salary{};
};
```

После имени нашего класса мы поставили двоеточие и через модификатор доступа `public` написали имя класса, от которого хотим произвести наследование. Теперь в нашем дочернем классе, либо как его ещё называют в производном классе, имеется доступ ко всем свойствам и методам класса родителя (потомка, или по-другому – суперкласса), которые являются публичными.

В начало

Модификаторы доступа при наследовании

При наследовании свойств класса мы указали модификатор `public`, это означает, что все свойства и методы суперкласса (класса потомка), помеченные этим модификатором, будут доступны в нашем классе. Нужно заметить, что свойства и методы, помеченные в суперклассе, как `private`, так и останутся закрытыми для нас.

Существует возможность при наследовании указывать и два других модификатора: `private` и `protected`. Если мы укажем при наследовании приватный модификатор, то мы искусственно для себя закроем все члены родительского класса. Таким образом, мы будем являться производным классом, но взаимодействовать с корневым классом мы не сможем. Это может потребоваться, если нам нужно просто реализовать какой-то интерфейс, либо создать иерархическую зависимость сущностей, об этом чуть позже.

Модификатор доступа `protected`, как было указано в прошлой работе служит, как раз для наследования. *Поля и методы, помеченные этим модификатором, будут открыты, только для производных классов.* Для доступа через объекты они будут закрыты. Если мы наследуем класс через

модификатор `protected`, то все поля и методы наследуемого класса, помеченные этим модификатором, либо модификатором `public` для нас искусственно станут `protected`. Здесь стоит обратить внимание на связи между модификаторами доступа наследования и модификаторами доступа членов объекта:

- Если мы наследуемся через `public`, то нам доступны все члены суперкласса, кроме тех, которые помечены – `private`.
- Если мы наследуемся через `private`, то мы закрываем для себя доступ ко всем членам класса, независимо от их модификатора.
- Если мы наследуемся через модификатор `protected`, то все члены суперкласса, которые не являются `private`, становятся `protected`.

Давайте перепишем прошлый пример используя модификатор `protected`:

```
class Human {

public:
	typedef char* string;
	typedef unsigned int score;

	enum class Gender {
		male,
		female
	};

	string getName() {
		return m_name;
	}
	score getAge() {
		return m_age;
	}
	Gender getGender() {
		return m_gender;
	}

	void setName() {
		/* реализация */
	}
	void setAge() {
		/* реализация */
	}
	void setGender() {
		/* реализация */
	}

protected:
	string m_name{};
	score m_age{};
	Gender m_gender{};
};
```

Теперь все поля сущности человек, доступны нам в классе банковского работника. Но мы не сможем к ним обратиться через объект класса.

В начало

Способы инициализации объекта

В прошлой работе было описано создание объекта класса и его инициализации. Показанная инициализация объекта является не эффективной и сложной в понимании, что происходит внутри. Давайте рассмотрим другие способы инициализации.

Создадим простые классы разных типов и попробуем создать от них объекты разными способами:

```
class T_1 {
public:
	int m_a{}, m_b{}, m_c{};
};

class T_2 {
public:
	int m_a{}, m_b{}, m_c{};

	T_2() {
		m_a = m_b = m_c = 5;
	}
};

class T_3 {
public:
	int m_a{}, m_b{}, m_c{};

	T_3(int n) {
		m_a = m_b = m_c = n;
	}
};
```

У нас создано три класса, в первом у нас нет конструктора, во – втором у нас создан конструктор без аргументов (такие конструкторы ещё называют конструкторами по умолчанию), в – третьем у нас конструктор принимает один аргумент. Все эти классы будут предоставлять различные способы инициализации своих объектов. Тема инициализации переменных весьма сложна и будет не понятна на данном этапе, поэтому мы рассмотрим лишь базовые принципы.

```
// простая инициализация объекта
// вызывается конструктор по умолчанию,
// т.к. мы явно в классе задали начальное
// значение для полей
// они будут иметь это значение
T_1 obj_1;

// так делать нельзя, это будет распознано
// словно мы хотим создать прототип функции
// который возвращает тип T_1
T_1 obj_2();

/*
	Однако, если класс имеет конструктор
```

```
	который принимает какие-то значения,
	то такая инициализация допускается
	и считается самой эффективной
*/

// мы создали объект класса, у которого
// конструктор принимает один аргумент
// при вызове конструктора происходит присваивание
// значений полям
T_3 obj_3(1);

/*
	Если конструктор явно создан и не принимает аргументов,
	то для эффективного создания объекта мы должны использовать
	первый способ создания:
*/

// здесь явно будет вызван конструктор класса
T_2 obj_4;

// эта форма называется формой uniform-инициализации
// эта универсальная форма инициализации была введена
// в 11-ом стандарте языка и позволяет одинаково
// для всех объектов выполнить инициализацию
// но на самом деле, под капотом всё куда сложнее

// мы создаём объект и применяем форму инициализации
// при таком подходе конструктор не вызывается,
// а объект инициализируется значениями из списка, переданного ему
T_1 obj_5{ 1, 2, 3 };

// тоже самое, что и выше
T_1 obj_6 = { 1, 2, 3 };

/*
	От универсальной инициализации объекта стоит отличать другую форму
	при которой используется равно:
*/

// для простых объектов это не имеет значения,
// но для пользовательских типов есть огромная разница
// речь идёт о копирующей инициализации
// самой затратной в плане ресурсов
// при ней создаётся анонимный объект указанного типа,
// а затем, он копируется в созданный объект слева
// после чего, анонимный объект удаляется
// создание двух объектов, их копирование и удаление
// очень затратная операция, поэтому такая инициализация применяется
// в крайней необходимости
	T_1 obj_7 = T_1{ 1, 2, 3 };

//тоже самое, что и выше
T_2 obj_8 = T_2();
/*
	Также, стоит обратить внимание, что следующая инициализация
	не является копирующей и её мы можем спокойно использовать
	не боясь, что упустим производительность
*/
// создание ссылки на объект
T_2* p_obj = new T_2();
```

В начало

Виртуальные функции и абстрактные классы

Продолжим разговор о наследовании. Предположим, мы хотим написать игру «Рисование геометрических фигур», что нам для этого может понадобиться? Сначала разберёмся, что такое геометрическая фигура? Это объект, у которого есть определённые свойства. Например, квадрат – у него четыре стороны, каждая сторона имеет длину, а он сам имеет цвет. Это свойства геометрической фигуры типа – многоугольник. Также, эти фигуры можно вращать, смещать и отрисовывать на экране – это их поведение. Чтобы не писать лишний код мы заметим, что все фигуры, с которыми мы будем работать, будут иметь одно и то же поведение – отрисовываться на экране, а также поворачиваться в разные стороны и смещаться. Получается, нам нужен какой-то базовый класс фигуры, в котором будут реализованы эти методы, но есть несколько проблем:

1. От какой фигуры мы можем унаследовать все остальные?
2. Не все фигуры одинаково отрисовываются и смещаются, для каждой нужна своя реализация метода.

Попробуем решить первый вопрос. Скажем, унаследуемся от класса многоугольник, но тогда мы не сможем унаследовать от него класс «Окружность», ведь окружность это такой себе многоугольник. Что делать? Выход есть, нужно создать некий базовый класс, который не будет описывать конкретные свойства фигур, т е не будет иметь полей, но будет иметь базовые методы работы с этими полями. Такой класс называется *абстрактным классом*. Давайте его напишем:

```
class Shape {

public:
	char* m_type_shape{};
	int scale{};

	void draw() {
		cout << "Рисуем фигуру" << endl;
	}

	void rotation() {
		cout << "Крутим фигуру" << endl;
	}

	void motion() {
		cout << "Перемещаем фигуру" << endl;
	}
};
```

Как мы видим, нас абстрактный класс описывает и реализует основные методы фигуры, а также описывает основные поля. Абстрактный класс, также может и вовсе не иметь реализации методов, а просто описывать их:

```
class Shape {

public:
	char* m_type_shape{};
	int scale{};

	void draw();

	void rotation();

	void motion();
};
```

Поэтому от абстрактного класса нельзя создать объект. Его обязательно нужно унаследовать и только тогда, уже от потомка создать объект. Что делать с нереализованными функциями? Всё просто их нужно переопределить в классах потомках. Но стандартное переопределение функций не годится для методов. Здесь используется особый спецификатор: `virtual`, который позволяет создавать виртуальные функции. Что такое виртуальная функция? *Виртуальная функция* – это метод, который при вызове у объекта использует свою реализацию, которая ближе всего к вызываемому объекту. Иными словами, если мы такую функцию перепишем в дочернем классе, то при вызове её у объекта дочернего класса будет выполнена реализация, которая прописана в этом объекте. Такое поведение реализует последний принцип ООП – **полиморфизм**. Перепишем методы в виртуальные функции:

```
class Shape {

public:
	char* m_type_shape{};
	int scale{};

	virtual void draw();

	virtual void rotation();

	virtual void motion();
};
```

Теперь наши функции являются виртуальными, а это значит мы их можем переопределить в дочерних классах. Но, есть большой недочёт в нашем коде. Сейчас мы всё ещё можем создать объект класса `Shape`, но мы должны запретить подобное поведение и сказать явно компилятору, что он имеет дело с абстрактным классом. Чтобы класс стал абстрактным, он должен иметь хотя бы одну чистую виртуальную функцию. *Чистая виртуальная функция* выглядит так:

```
virtual void draw() = 0;
```

Перепишем наш пример ещё раз:

```
class Shape {
```

```
public:
	char* m_type_shape{};
	int scale{};

	virtual void draw() = 0;

	virtual void rotation() = 0;

	virtual void motion() = 0;
};
```

Теперь мы явно сообщили компилятору, что наш класс является абстрактным, и что все его методы должны быть переопределены в дочернем классе. Давайте унаследуем от него какой-то класс фигуры, скажем квадрат:

```
class Square : public Shape {

public:
	virtual void draw() {
		cout << "Рисуем квадрат" << endl;
	}

	virtual void rotation() {
		cout << "Крутим квадрат" << endl;
	}

	virtual void motion() {
		cout << "Перемещаем квадрат" << endl;
	}

};
```

Теперь, при создании объекта этого класса, мы получим переопределённое поведение методов.

В начало

Коротко об интерфейсах

Глядя на нашу реализацию, хочется невольно спросить, а зачем мы всё это проделывали, если всё равно пришлось реализовывать в каждом классе свой метод отрисовки? На самом деле мы сделали большое дело для реализации принципа полиморфизма. Давайте напишем функцию, которая будет определять тип фигуры и отрисовывать её на экране. Она может выглядеть как – то так:

```
void function(Square &sq) {
	cout << sq.m_type_shape << endl;
	sq.draw();
}
```

(обратите внимание, каким образом мы передаём сюда объект)

Функция принимает объект класса квадрат, печатает этот тип и вызывает функцию отрисовки. Но что, если мы захотим написать такую функцию для

окружности, прямоугольника, ромба, овала? Нам придётся делать множество перегрузок функций с одним и тем же телом. Такой подход весьма неэффективен.

Вспоминая идею полиморфизма, мы хотим написать функцию, принимающую любой объект, который обладает таким интерфейсом, с которым эта функция умеет работать. Абстрактный класс фигуры в качестве интерфейса волне подойдёт. Перепишем наш пример, но сначала для наглядности добавим класс `Circle`:

```
class Circle : public Shape {

public:
	virtual void draw() {
		cout << "Рисуем окружность" << endl;
	}

	virtual void rotation() {
		cout << "Крутим окружность" << endl;
	}

	virtual void motion() {
		cout << "Перемещаем окружность" << endl;
	}

};
int main() {

	Circle circle;
	Square square;

	function(circle);
	function(square);

	return 0;
}
```

Мы создали два объекта разных классов, но смогли передать их в одну функцию, которая, зная интерфейс этих объектов умело их обработала. Мы добились чего хотели. Осталось разобраться ещё в одном термине – интерфейс. *Интерфейс* в языке C++ это класс, у которого есть только чистые виртуальные функции, реализации их он не имеет и полей у него тоже нет. Такие интерфейсы созданы для того, чтобы обрабатывать общее поведение объектов, которые их наследуют. Когда класс наследует интерфейс, говорят, что он реализует его. В отличии от наследования абстрактного класса, методы реализуемого интерфейса должны быть все переопределены в дочернем классе.

В начало

Реализация по умолчанию чистых виртуальных функций

В абстрактных классах можно определять чистые виртуальные функции, это значит, что мы можем задать реализацию по умолчанию, которую дочерний класс может и не менять. Для того, чтобы это сделать необходимо обязательно вынести реализацию из класса, рассмотрим пример:

```
class Shape {

public:
	char* m_type_shape{};
	int scale{};

	virtual void draw() = 0;

	virtual void rotation() = 0;

	virtual void motion() = 0;
};

void Shape::draw() {
	cout << "Рисуем фигуру" << endl;
}
```

`Draw` – по-прежнему чистая виртуальная функция, только теперь у неё есть реализация по умолчанию. Чтобы использовать эту реализацию в дочернем классе, нужно сделать следующее:

```
class CertainShape : public Shape {

public:
	virtual void draw() {
		Shape::draw();
	}
};
```

Как видим, в переопределении функции мы вызываем её у класса `Shape` через оператор разрешения области видимости.

Стоит отметить, что не следует объявлять все методы виртуальными, виртуальные функции имеют сложный механизм вызова, поэтому уступают по скорости обычным функциям. Делайте это только там, где это действительно нужно.

Модификаторы override и final

Override – явно указывает на то, что виртуальная функция должна быть переопределена в классе родителе.

Final – явно указывает на то, что виртуальная функция не должна быть переопределена в классе родителе. Если этот модификатор указать после имени класса, то такой класс нельзя будет наследовать.

В начало

Введение в UML

Унифицированный язык моделирования (UML) является стандартным инструментом для создания «чертежей» программного обеспечения. С помощью UML можно визуализировать, специфицировать, конструировать и документировать артефакты программных систем.

UML пригоден для моделирования любых систем: от информационных систем масштаба предприятия до распределенных Web-приложений и даже встроенных систем реального времени. Это очень выразительный язык, позволяющий рассмотреть систему со всех точек зрения, имеющих отношение к её разработке и последующему развертыванию. Несмотря на обилие выразительных возможностей, этот язык прост для понимания и использования. Изучение UML удобнее всего начать с его концептуальной модели, которая включает в себя три основных элемента: базовые строительные блоки, правила, определяющие, как эти блоки могут сочетаться между собой, и некоторые общие механизмы языка.

Несмотря на свои достоинства, UML – это всего лишь язык; он является одной из составляющих процесса разработки программного обеспечения, и не более того. Хотя UML не зависит от моделируемой реальности, лучше всего применять его, когда процесс моделирования основан на рассмотрении прецедентов использования, является итеративным и пошаговым, а сама система имеет четко выраженную архитектуру.

UML – это язык для визуализации, специфицирования, конструирования и документирования артефактов программных систем.

Язык состоит из словаря и правил, позволяющих комбинировать входящие в него слова и получать осмысленные конструкции. В языке моделирования словарь и правила ориентированы на концептуальное и физическое представление системы. Язык моделирования, подобный UML, является стандартным средством для составления «чертежей» программного обеспечения.

Моделирование необходимо для понимания системы. При этом единственной модели никогда не бывает достаточно. Напротив, для понимания любой нетривиальной системы приходится разрабатывать большое количество взаимосвязанных моделей. В применении к программным системам это означает, что необходим язык, с помощью которого можно с различных точек зрения описать представления архитектуры системы на протяжении цикла ее разработки.

Концептуальная модель UML

Словарь языка UML включает три вида строительных блоков:

- сущности;

- отношения;
- диаграммы.

Сущности – это абстракции, являющиеся основными элементами модели. Отношения связывают различные сущности; диаграммы группируют представляющие интерес совокупности сущностей.

В UML имеется четыре типа сущностей:

- структурные;
- поведенческие;
- группирующие;
- аннотационные.

Сущности являются основными объектно-ориентированными блоками языка. С их помощью можно создавать корректные модели.

Структурные сущности – это имена существительные в моделях на языке UML. Как правило, они представляют собой статические части модели, соответствующие концептуальным или физическим элементам системы. Одной из разновидностей структурных сущностей является сущность класс.

Класс (Class) – это описание совокупности объектов с общими атрибутами, операциями, отношениями и семантикой. Класс реализует один или несколько интерфейсов. Графически класс изображается в виде прямоугольника, в котором обычно записаны его имя, атрибуты и операции, как показано на рисунке 1.

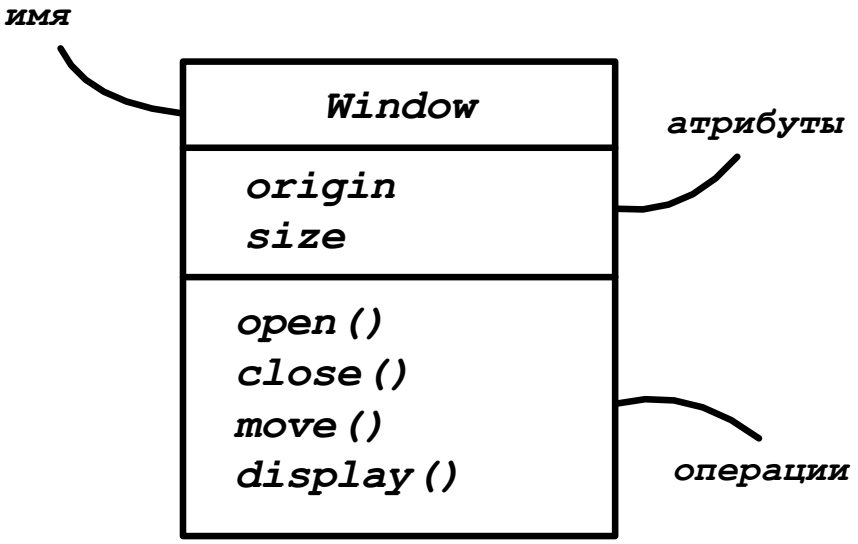


Рис. 1. Класс «Window»

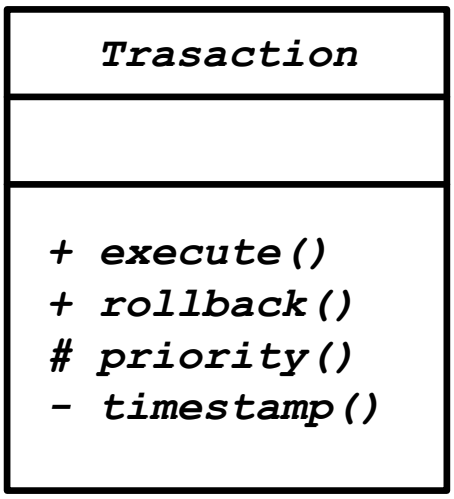


Рис. 2. Абстрактный класс «Transaction»

Обратите внимание, что модификаторы доступа в сущностях «класс» отображаются следующим образом:

\+ - `public`;
– - `private`;
- `protected`.

В языке UML определены четыре типа отношений:

- зависимость;
- ассоциация;
- обобщение;
- реализация.

Эти отношения являются основными связующими строительными блоками в UML и применяются для создания корректных моделей.

Зависимость (Dependency) – это семантическое отношение между двумя сущностями, при котором изменение одной из них, независимой, может повлиять на семантику другой, зависимой. Графически зависимость изображается в виде прямой пунктирной линии, часто со стрелкой, которая может содержать метку.

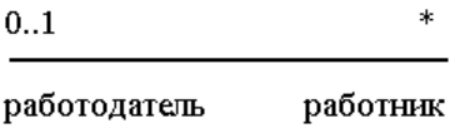
Зависимость

Ассоциация (Association) – структурное отношение, описывающее совокупность связей; связь – это соединение между объектами.

Ассоциация

Разновидностью ассоциации является *агрегирование* (Aggregation) – так называют структурное отношение между целым и его частями. Графически

ассоциация изображается в виде прямой линии (иногда завершающейся стрелкой или содержащей метку), рядом с которой могут присутствовать дополнительные обозначения, например кратность и имена ролей.



Агрегация – особая разновидность ассоциации, представляющая структурную связь целого с его частями. Как тип ассоциации, агрегация может быть именованной. Одно отношение агрегации не может включать более двух классов (контейнер и содержимое).

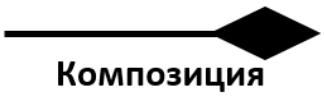
Агрегация встречается, когда один класс является коллекцией или контейнером других. Причём, по умолчанию агрегацией называют агрегацию по ссылке, то есть, когда время существования содержащихся классов не зависит от времени существования содержащего их класса. Если контейнер будет уничтожен, то его содержимое – нет.

Графически агрегация представляется пустым ромбом на блоке класса «целое», и линией, идущей от этого ромба к классу «часть».

Композиция – более строгий вариант агрегации. Известна также как агрегация по значению.

Композиция – это форма агрегации с четко выраженными отношениями владения и совпадением времени жизни частей и целого. Композиция имеет жёсткую зависимость времени существования экземпляров класса контейнера и экземпляров содержащихся классов. Если контейнер будет уничтожен, то всё его содержимое будет также уничтожено.

Графически представляется как и агрегация, но с закрашенным ромбиком.



Обобщение (Generalization) – это отношение «специализация/обобщение», при котором объект специализированного элемента (потомок) может быть подставлен вместо объекта обобщенного элемента (родителя или предка). Таким образом, потомок (Child) наследует структуру и поведение своего родителя (Parent). Графически отношение обобщения изображается в виде линии с незакрашенной стрелкой, указывающей на родителя.

Реализация (Realization) – это семантическое отношение между классификаторами, при котором один классификатор определяет «контракт», а другой гарантирует его выполнение.

Отношения реализации встречаются в двух случаях: во-первых, между интерфейсами и реализующими их классами или компонентами, во-вторых, между прецедентами и реализующими их кооперациями. Отношение реализации изображается в виде пунктирной линии с незакрашенной стрелкой, как нечто среднее между отношениями обобщения и зависимости.

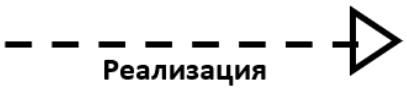


Диаграмма в UML – это графическое представление набора элементов, изображаемое чаще всего в виде связанного графа с вершинами (сущностями) и ребрами (отношениями). Диаграммы рисуют для визуализации системы с разных точек зрения. Диаграмма – в некотором смысле одна из проекций системы. Как правило, за исключением наиболее тривиальных случаев, диаграммы дают свернутое представление элементов, из которых составлена система. Один и тот же элемент может присутствовать во всех диаграммах, или только в нескольких (самый распространенный вариант), или не присутствовать ни в одной (очень редко). Теоретически диаграммы могут содержать любые комбинации сущностей и отношений. На практике, однако, применяется сравнительно небольшое количество типовых комбинаций, соответствующих пяти наиболее употребительным видам, которые составляют архитектуру программной системы.

Таким образом, в UML выделяют девять основных типов диаграмм:

- диаграммы классов;
- диаграммы объектов;
- диаграммы прецедентов;
- диаграммы последовательностей;
- диаграммы кооперации;
- диаграммы состояний;
- диаграммы действий;
- диаграммы компонентов;
- диаграммы развертывания.

На *диаграмме классов* показывают классы, интерфейсы, объекты и кооперации, а также их отношения. При моделировании объектно-ориентированных систем этот тип диаграмм используют чаще всего. Диаграммы классов соответствуют статическому виду системы с точки зрения проектирования (см. Рис. 3).

На *диаграмме объектов* представлены объекты и отношения между ними. Они являются статическими «фотографиями» экземпляров сущностей,

показанных на диаграммах классов. Диаграммы объектов, как и диаграммы классов, относятся к статическому виду системы с точки зрения проектирования или процессов, но с расчетом на настоящую или макетную реализацию.

На *диаграмме прецедентов* представлены прецеденты и актеры (частный случай классов), а также отношения между ними. Диаграммы прецедентов относятся к статическому виду системы с точки зрения прецедентов использования. Они особенно важны при организации и моделировании поведения системы.

Диаграммы последовательностей и кооперации являются частными случаями диаграмм взаимодействия. На диаграммах взаимодействия представлены связи между объектами; показаны, в частности, сообщения, которыми объекты могут обмениваться. Диаграммы взаимодействия относятся к динамическому виду системы. При этом диаграммы последовательности отражают временную упорядоченность сообщений, а диаграммы кооперации – структурную организацию обменивающихся сообщениями объектов. Эти диаграммы являются изоморфными, то есть могут быть преобразованы друг в друга.

На *диаграммах состояний* (Statechart diagrams) представлен автомат, включающий в себя состояния, переходы, события и виды действий. Диаграммы состояний относятся к динамическому виду системы; особенно они важны при моделировании поведения интерфейса, класса или кооперации. Они акцентируют внимание на поведении объекта, зависящем от последовательности событий, что очень полезно для моделирования реактивных систем.

Диаграмма деятельности – это частный случай диаграммы состояний; на ней представлены переходы потока управления от одной деятельности к другой внутри системы. Диаграммы деятельности относятся к динамическому виду системы; они наиболее важны при моделировании ее функционирования и отражают поток управления между объектами.

На *диаграмме компонентов* представлены организация совокупности компонентов и существующие между ними зависимости. Диаграммы компонентов относятся к статическому виду системы с точки зрения реализации. Они могут быть соотнесены с диаграммами классов, так как компонент обычно отображается на один или несколько классов, интерфейсов или кооперацией.

На *диаграмме развертывания* представлена конфигурация обрабатывающих узлов системы и размещенных в них компонентов. Диаграммы развертывания относятся к статическому виду архитектуры системы с точки зрения развертывания. Они связаны с диаграммами компонентов, поскольку в узле обычно размещаются один или несколько компонентов.

При графическом построении диаграммы необходимо выполнение следующих правил:

- имя диаграммы по возможности должно соответствовать ее назначению;
- элементы должны быть расположены так, чтобы число пересечений между ними было минимальным;
- элементы, выражающие семантически близкие сущности, должны располагаться на диаграмме рядом;
- при необходимости концентрации внимания на важной части диаграммы допускается ее выделение цветом.

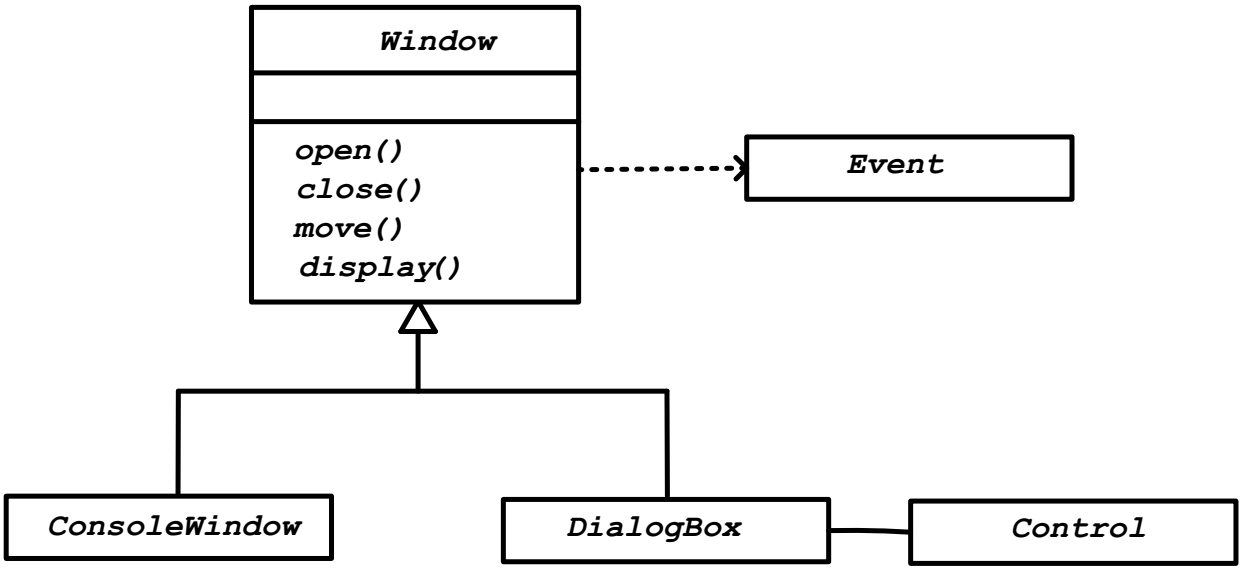


Рис. 3. Пример диаграммы классов

В начало

Общее задание

В следующем цикле лабораторных работ по курсу «Высокоуровневое программирование» Вам будет предложено реализовать полноценную программу по управлению, каталогизации и обработки информации. Каждая программа будет представлять из себя логически законченную платформу по типу: «Автоматизированная система управления» и содержать в себе следующий функционал:

- Обработка базы данных
- Создание / удаление / редактирования записей
- Сортировка и фильтрация записей

Управление пользователями

- Создание
- Удаление
- Авторизация

Каждая программа будет снабжена интерактивным меню пользователя.

В этой лабораторной работе Вам будет предложено реализовать такое меню, а также создать файл с функциями-«заглушками» для тестирования меню. В каждом варианте задания будет описана функциональность программы, которую Вы должны реализовать в меню.

Вариант №1

Общая задача

Вам будет предложено написать программу – «Автоматизированная система института». Которая будет включать следующий функционал:

- Ведение базы студентов и преподавателей (разные базы)
  - Создание / удаление / редактирование записей
  - Сортировка / фильтрация
- Ведение базы предметов
- Возможность авторизации
- Создание файлов-отчётов и сохранения состояния

Базовая часть

Задача 1

Создайте новый проект в студии, или создайте новую директорию, в которой будут храниться исходные файлы вашей программы. В этой директории/проекте создайте папку под названием: «MyConMenu». Это будет ваша рабочая директория, в которой будут лежать исходные файлы библиотеки. В корне создайте файл: «CMenu.cpp» и «CMenu.h».

Задача 2

Реализовывать функционал меню Вы будете в новом пространстве имён, которое будет изолировать ваши имена объектов от сущностей из других библиотек. Для начала создайте новое пространство имён в файле заголовка и в файле исходного кода. Название этого пространства имён будет включать ваши инициалы, пример: ФИО.

Задача 3

В заголовочном файле в новом пространстве имён создайте класс: CMenu, который будет включать следующие закрытые поля: поле типа: int «m_select», которое будет содержать последний ввод пользователя, по умолчанию равняется: -1, поле типа: bool «m_running», которое будет отвечать за выполнение меню, по умолчанию: false, поле типа: char* «m_title», которое будет содержать ссылку на С-строку, которая будет хранить заголовок вашего меню, по умолчанию нулевой указатель и поле типа: size_t «m_count», которое будет хранить количество пунктов меню.

Задача 4

Ваше меню, будет принимать в себя заголовок и массив классов, каждый из которых будет представлять обёртку для запуска нужной функции. Теперь давайте создадим абстрактный класс, от которого впоследствии будем наследовать другие классы с пунктом меню. Для этого в директории «MyConMenu», создайте два новых файла: «AbstractItemMenu.cpp» и «AbstractItemMenu.h». В этих файлах пропишите своё пространство имён. В заголовочном файле, в пространстве имён, создайте класс: «ItemMenu». В этом классе реализуйте следующие закрытые поля: «m_item_name» типа char*, который будет содержать указатель на С-строку названия пункта, по умолчанию – нулевой указатель. И поле: «m_func» типа Func, это поле будет содержать указатель на запускаемую функцию. По умолчанию нулевой указатель.

Задача 5

В классе ItemMenu создайте новую обёртку для прототипа функции через typedef, которая будет называться Func и являться псевдонимом типа: int(*)(). Этот тип будет описывать указатель на функцию, которая ничего не принимает и возвращает значение типа: int. Теперь реализуйте конструктор ItemMenu. Который будет принимать указатель на С-строку и указатель на функцию. Реализацию конструктора выполните в файле исходного кода. Также, создайте виртуальный геттер для поля названия пункта. И для него же создайте виртуальную функцию, которая будет печатать этот пункт в консоли. Реализацию виртуальных функций выполните в файле исходного кода. Также, создайте виртуальную функцию run(), которая будет запускать функцию, которая хранится в поле класса. Данная виртуальная функция будет возвращать значение, которое вернула запускаемая функция.

Задача 6

В заголовочном файле, в классе CMenu добавьте в закрытые поля поле, которое будет хранить указатель на наш массив объектов нашего абстрактного класса: ItemMenu, назовите это поле: m_items. По умолчанию нулевой указатель. Теперь реализуйте конструктор для меню, который будет принимать название меню, массив объектов типа элемента меню и их количество. Реализацию также вынесете в файл исходного кода. Также, реализуйте геттеры для всех полей класса.

Задача 7

Последним заданием в этой Л/Р будет создание двух методов класса меню: метода печати всех пунктов на экран (print) и метода считывания пользовательского ввода (runCommand). Последний метод не должен пока обрабатывать корректность этого ввода, данная функциональность будет рассмотрена в другой работе. Также, метод считывания пользовательского ввода, должен возвращать результат выполнения функции, которую он запускает. Полная функциональность его будет следующей: метод запускает

функцию вывода на экран всех пунктов и приглашает пользователя сделать ввод. После ввода, он находит в массиве элементов нужный пункт по индексу и запускает метод run(). Возвращаемое значение этого метода, он также возвращает.

Индивидуальная часть

Задача 1

Протестируйте созданное пользовательское меню. Создайте классы наследники, которые будут имплементировать методы абстрактного класса: ItemMenu. Для этого реализуйте функции-«заглушки», которые впоследствии будут выполнять логику вашего задания. Подумайте, какие перегрузки можно сделать для виртуальных функций этого класса? Реализуйте эти перегрузки.

Задача 2

В следующей Л/Р Вы начнёте реализовывать непосредственно свою программу. Для начала подумайте над структурой будущего программного продукта. Разбейте его на смысловые блоки и модули. Создайте эту иерархию представлений в виде папок и классов.

В начало

Вариант №2

Общая задача

Вам будет предложено написать программу – «Автоматизированная система школы». Которая будет включать следующий функционал:

- Ведение базы школьников и учителей (разные базы)
  - Создание / удаление / редактирование записей
  - Сортировка / фильтрация
- Ведение базы предметов
- Возможность авторизации
- Создание файлов-отчётов и сохранения состояния

Базовая часть

Задача 1

Создайте новый проект в студии, или создайте новую директорию, в которой будут храниться исходные файлы вашей программы. В этой директории/проекте создайте папку под названием: «MyConMenu». Это будет ваша рабочая директория, в которой будут лежать исходные файлы библиотеки. В корне создайте файл: «CMenu.cpp» и «CMenu.h».

Задача 2

Реализовывать функционал меню Вы будете в новом пространстве имён, которое будет изолировать ваши имена объектов от сущностей из других библиотек. Для начала создайте новое пространство имён в файле заголовка и

в файле исходного кода. Название этого пространства имён будет включать ваши инициалы, пример: ФИО.

Задача 3

В заголовочном файле в новом пространстве имён создайте класс: CMenu, который будет включать следующие закрытые поля: поле типа: int «m_select», которое будет содержать последний ввод пользователя, по умолчанию равняется: -1, поле типа: bool «m_running», которое будет отвечать за выполнение меню, по умолчанию: false, поле типа: char* «m_title», которое будет содержать ссылку на С-строку, которая будет хранить заголовок вашего меню, по умолчанию нулевой указатель и поле типа: size_t «m_count», которое будет хранить количество пунктов меню.

Задача 4

Ваше меню, будет принимать в себя заголовок и массив классов, каждый из которых будет представлять обёртку для запуска нужной функции. Теперь давайте создадим абстрактный класс, от которого впоследствии будем наследовать другие классы с пунктом меню. Для этого в директории «MyConMenu», создайте два новых файла: «AbstractItemMenu.cpp» и «AbstractItemMenu.h». В этих файлах пропишите своё пространство имён. В заголовочном файле, в пространстве имён, создайте класс: «ItemMenu». В этом классе реализуйте следующие закрытые поля: «m_item_name» типа char*, который будет содержать указатель на С-строку названия пункта, по умолчанию – нулевой указатель. И поле: «m_func» типа Func, это поле будет содержать указатель на запускаемую функцию. По умолчанию нулевой указатель.

Задача 5

В классе ItemMenu создайте новую обёртку для прототипа функции через typedef, которая будет называться Func и являться псевдонимом типа: int(*)(). Этот тип будет описывать указатель на функцию, которая ничего не принимает и возвращает значение типа: int. Теперь реализуйте конструктор ItemMenu. Который будет принимать указатель на С-строку и указатель на функцию. Реализацию конструктора выполните в файле исходного кода. Также, создайте виртуальный геттер для поля названия пункта. И для него же создайте виртуальную функцию, которая будет печатать этот пункт в консоли. Реализацию виртуальных функций выполните в файле исходного кода. Также, создайте виртуальную функцию run(), которая будет запускать функцию, которая хранится в поле класса. Данная виртуальная функция будет возвращать значение, которое вернула запускаемая функция.

Задача 6

В заголовочном файле, в классе CMenu добавьте в закрытые поля поле, которое будет хранить указатель на наш массив объектов нашего абстрактного класса: ItemMenu, назовите это поле: m_items. По умолчанию нулевой

указатель. Теперь реализуйте конструктор для меню, который будет принимать название меню, массив объектов типа элемента меню и их количество. Реализацию также вынесете в файл исходного кода. Также, реализуйте геттеры для всех полей класса.

Задача 7

Последним заданием в этой Л/Р будет создание двух методов класса меню: метода печати всех пунктов на экран (print) и метода считывания пользовательского ввода (runCommand). Последний метод не должен пока обрабатывать корректность этого ввода, данная функциональность будет рассмотрена в другой работе. Также, метод считывания пользовательского ввода, должен возвращать результат выполнения функции, которую он запускает. Полная функциональность его будет следующей: метод запускает функцию вывода на экран всех пунктов и приглашает пользователя сделать ввод. После ввода, он находит в массиве элементов нужный пункт по индексу и запускает метод run(). Возвращаемое значение этого метода, он также возвращает.

Индивидуальная часть

Задача 1

Протестируйте созданное пользовательское меню. Создайте классы наследники, которые будут имплементировать методы абстрактного класса: ItemMenu. Для этого реализуйте функции-«заглушки», которые впоследствии будут выполнять логику вашего задания. Подумайте, какие перегрузки можно сделать для виртуальных функций этого класса? Реализуйте эти перегрузки.

Задача 2

В следующей Л/Р Вы начнёте реализовывать непосредственно свою программу. Для начала подумайте над структурой будущего программного продукта. Разбейте его на смысловые блоки и модули. Создайте эту иерархию представлений в виде папок и классов.

В начало

Вариант №3

Общая задача

Вам будет предложено написать программу – «Автоматизированная система парикмахерской». Которая будет включать следующий функционал:

- Ведение базы сотрудников и клиентов (разные базы)
  - Создание / удаление / редактирование записей
  - Сортировка / фильтрация
- Ведение базы услуг
- Возможность авторизации
- Создание файлов-отчётов и сохранения состояния

Базовая часть

Задача 1

Создайте новый проект в студии, или создайте новую директорию, в которой будут храниться исходные файлы вашей программы. В этой директории/проекте создайте папку под названием: «MyConMenu». Это будет ваша рабочая директория, в которой будут лежать исходные файлы библиотеки. В корне создайте файл: «CMenu.cpp» и «CMenu.h».

Задача 2

Реализовывать функционал меню Вы будете в новом пространстве имён, которое будет изолировать ваши имена объектов от сущностей из других библиотек. Для начала создайте новое пространство имён в файле заголовка и в файле исходного кода. Название этого пространства имён будет включать ваши инициалы, пример: ФИО.

Задача 3

В заголовочном файле в новом пространстве имён создайте класс: CMenu, который будет включать следующие закрытые поля: поле типа: int «m_select», которое будет содержать последний ввод пользователя, по умолчанию равняется: -1, поле типа: bool «m_running», которое будет отвечать за выполнение меню, по умолчанию: false, поле типа: char* «m_title», которое будет содержать ссылку на С-строку, которая будет хранить заголовок вашего меню, по умолчанию нулевой указатель и поле типа: size_t «m_count», которое будет хранить количество пунктов меню.

Задача 4

Ваше меню, будет принимать в себя заголовок и массив классов, каждый из которых будет представлять обёртку для запуска нужной функции. Теперь давайте создадим абстрактный класс, от которого впоследствии будем наследовать другие классы с пунктом меню. Для этого в директории «MyConMenu», создайте два новых файла: «AbstractItemMenu.cpp» и «AbstractItemMenu.h». В этих файлах пропишите своё пространство имён. В заголовочном файле, в пространстве имён, создайте класс: «ItemMenu». В этом классе реализуйте следующие закрытые поля: «m_item_name» типа char*, который будет содержать указатель на С-строку названия пункта, по умолчанию – нулевой указатель. И поле: «m_func» типа Func, это поле будет содержать указатель на запускаемую функцию. По умолчанию нулевой указатель.

Задача 5

В классе ItemMenu создайте новую обёртку для прототипа функции через typedef, которая будет называться Func и являться псевдонимом типа: int(*)(). Этот тип будет описывать указатель на функцию, которая ничего не принимает и возвращает значение типа: int. Теперь реализуйте конструктор ItemMenu. Который будет принимать указатель на С-строку и указатель на

функцию. Реализацию конструктора выполните в файле исходного кода. Также, создайте виртуальный геттер для поля названия пункта. И для него же создайте виртуальную функцию, которая будет печатать этот пункт в консоли. Реализацию виртуальных функций выполните в файле исходного кода. Также, создайте виртуальную функцию run(), которая будет запускать функцию, которая хранится в поле класса. Данная виртуальная функция будет возвращать значение, которое вернула запускаемая функция.

Задача 6

В заголовочном файле, в классе CMenu добавьте в закрытые поля поле, которое будет хранить указатель на наш массив объектов нашего абстрактного класса: ItemMenu, назовите это поле: m_items. По умолчанию нулевой указатель. Теперь реализуйте конструктор для меню, который будет принимать название меню, массив объектов типа элемента меню и их количество. Реализацию также вынесете в файл исходного кода. Также, реализуйте геттеры для всех полей класса.

Задача 7

Последним заданием в этой Л/Р будет создание двух методов класса меню: метода печати всех пунктов на экран (print) и метода считывания пользовательского ввода (runCommand). Последний метод не должен пока обрабатывать корректность этого ввода, данная функциональность будет рассмотрена в другой работе. Также, метод считывания пользовательского ввода, должен возвращать результат выполнения функции, которую он запускает. Полная функциональность его будет следующей: метод запускает функцию вывода на экран всех пунктов и приглашает пользователя сделать ввод. После ввода, он находит в массиве элементов нужный пункт по индексу и запускает метод run(). Возвращаемое значение этого метода, он также возвращает.

Индивидуальная часть

Задача 1

Протестируйте созданное пользовательское меню. Создайте классы наследники, которые будут имплементировать методы абстрактного класса: ItemMenu. Для этого реализуйте функции-«заглушки», которые впоследствии будут выполнять логику вашего задания. Подумайте, какие перегрузки можно сделать для виртуальных функций этого класса? Реализуйте эти перегрузки.

Задача 2

В следующей Л/Р Вы начнёте реализовывать непосредственно свою программу. Для начала подумайте над структурой будущего программного продукта. Разбейте его на смысловые блоки и модули. Создайте эту иерархию представлений в виде папок и классов.

В начало

Вариант №4

Общая задача

Вам будет предложено написать программу – «Автоматизированная система авто – салона». Которая будет включать следующий функционал:

- Ведение базы сотрудников и клиентов (разные базы)
  - Создание / удаление / редактирование записей
  - Сортировка / фильтрация
- Ведение базы автомобилей и услуг
- Возможность авторизации
- Создание файлов-отчётов и сохранения состояния

Базовая часть

Задача 1

Создайте новый проект в студии, или создайте новую директорию, в которой будут храниться исходные файлы вашей программы. В этой директории/проекте создайте папку под названием: «MyConMenu». Это будет ваша рабочая директория, в которой будут лежать исходные файлы библиотеки. В корне создайте файл: «CMenu.cpp» и «CMenu.h».

Задача 2

Реализовывать функционал меню Вы будете в новом пространстве имён, которое будет изолировать ваши имена объектов от сущностей из других библиотек. Для начала создайте новое пространство имён в файле заголовка и в файле исходного кода. Название этого пространства имён будет включать ваши инициалы, пример: ФИО.

Задача 3

В заголовочном файле в новом пространстве имён создайте класс: CMenu, который будет включать следующие закрытые поля: поле типа: int «m_select», которое будет содержать последний ввод пользователя, по умолчанию равняется: -1, поле типа: bool «m_running», которое будет отвечать за выполнение меню, по умолчанию: false, поле типа: char* «m_title», которое будет содержать ссылку на С-строку, которая будет хранить заголовок вашего меню, по умолчанию нулевой указатель и поле типа: size_t «m_count», которое будет хранить количество пунктов меню.

Задача 4

Ваше меню, будет принимать в себя заголовок и массив классов, каждый из которых будет представлять обёртку для запуска нужной функции. Теперь давайте создадим абстрактный класс, от которого впоследствии будем наследовать другие классы с пунктом меню. Для этого в директории «MyConMenu», создайте два новых файла: «AbstractItemMenu.cpp» и «AbstractItemMenu.h». В этих файлах пропишите своё пространство имён. В заголовочном файле, в пространстве имён, создайте класс: «ItemMenu». В этом

классе реализуйте следующие закрытые поля: «m_item_name» типа char*, который будет содержать указатель на С-строку названия пункта, по умолчанию – нулевой указатель. И поле: «m_func» типа Func, это поле будет содержать указатель на запускаемую функцию. По умолчанию нулевой указатель.

Задача 5

В классе ItemMenu создайте новую обёртку для прототипа функции через typedef, которая будет называться Func и являться псевдонимом типа: int(*)(). Этот тип будет описывать указатель на функцию, которая ничего не принимает и возвращает значение типа: int. Теперь реализуйте конструктор ItemMenu. Который будет принимать указатель на С-строку и указатель на функцию. Реализацию конструктора выполните в файле исходного кода. Также, создайте виртуальный геттер для поля названия пункта. И для него же создайте виртуальную функцию, которая будет печатать этот пункт в консоли. Реализацию виртуальных функций выполните в файле исходного кода. Также, создайте виртуальную функцию run(), которая будет запускать функцию, которая хранится в поле класса. Данная виртуальная функция будет возвращать значение, которое вернула запускаемая функция.

Задача 6

В заголовочном файле, в классе CMenu добавьте в закрытые поля поле, которое будет хранить указатель на наш массив объектов нашего абстрактного класса: ItemMenu, назовите это поле: m_items. По умолчанию нулевой указатель. Теперь реализуйте конструктор для меню, который будет принимать название меню, массив объектов типа элемента меню и их количество. Реализацию также вынесете в файл исходного кода. Также, реализуйте геттеры для всех полей класса.

Задача 7

Последним заданием в этой Л/Р будет создание двух методов класса меню: метода печати всех пунктов на экран (print) и метода считывания пользовательского ввода (runCommand). Последний метод не должен пока обрабатывать корректность этого ввода, данная функциональность будет рассмотрена в другой работе. Также, метод считывания пользовательского ввода, должен возвращать результат выполнения функции, которую он запускает. Полная функциональность его будет следующей: метод запускает функцию вывода на экран всех пунктов и приглашает пользователя сделать ввод. После ввода, он находит в массиве элементов нужный пункт по индексу и запускает метод run(). Возвращаемое значение этого метода, он также возвращает.

Индивидуальная часть

Задача 1

Протестируйте созданное пользовательское меню. Создайте классы наследники, которые будут имплементировать методы абстрактного класса: ItemMenu. Для этого реализуйте функции-«заглушки», которые впоследствии будут выполнять логику вашего задания. Подумайте, какие перегрузки можно сделать для виртуальных функций этого класса? Реализуйте эти перегрузки.

Задача 2

В следующей Л/Р Вы начнёте реализовывать непосредственно свою программу. Для начала подумайте над структурой будущего программного продукта. Разбейте его на смысловые блоки и модули. Создайте эту иерархию представлений в виде папок и классов.

В начало

Вариант №5

Общая задача

Вам будет предложено написать программу – «Автоматизированная система банка». Которая будет включать следующий функционал:

- Ведение базы сотрудников и клиентов (разные базы)
  - Создание / удаление / редактирование записей
  - Сортировка / фильтрация
- Ведение базы счетов
- Возможность авторизации
- Создание файлов-отчётов и сохранения состояния

Базовая часть

Задача 1

Создайте новый проект в студии, или создайте новую директорию, в которой будут храниться исходные файлы вашей программы. В этой директории/проекте создайте папку под названием: «MyConMenu». Это будет ваша рабочая директория, в которой будут лежать исходные файлы библиотеки. В корне создайте файл: «CMenu.cpp» и «CMenu.h».

Задача 2

Реализовывать функционал меню Вы будете в новом пространстве имён, которое будет изолировать ваши имена объектов от сущностей из других библиотек. Для начала создайте новое пространство имён в файле заголовка и в файле исходного кода. Название этого пространства имён будет включать ваши инициалы, пример: ФИО.

Задача 3

В заголовочном файле в новом пространстве имён создайте класс: CMenu, который будет включать следующие закрытые поля: поле типа: int «m_select», которое будет содержать последний ввод пользователя, по

умолчанию равняется: -1, поле типа: bool «m_running», которое будет отвечать за выполнение меню, по умолчанию: false, поле типа: char* «m_title», которое будет содержать ссылку на С-строку, которая будет хранить заголовок вашего меню, по умолчанию нулевой указатель и поле типа: size_t «m_count», которое будет хранить количество пунктов меню.

Задача 4

Ваше меню, будет принимать в себя заголовок и массив классов, каждый из которых будет представлять обёртку для запуска нужной функции. Теперь давайте создадим абстрактный класс, от которого впоследствии будем наследовать другие классы с пунктом меню. Для этого в директории «MyConMenu», создайте два новых файла: «AbstractItemMenu.cpp» и «AbstractItemMenu.h». В этих файлах пропишите своё пространство имён. В заголовочном файле, в пространстве имён, создайте класс: «ItemMenu». В этом классе реализуйте следующие закрытые поля: «m_item_name» типа char*, который будет содержать указатель на С-строку названия пункта, по умолчанию – нулевой указатель. И поле: «m_func» типа Func, это поле будет содержать указатель на запускаемую функцию. По умолчанию нулевой указатель.

Задача 5

В классе ItemMenu создайте новую обёртку для прототипа функции через typedef, которая будет называться Func и являться псевдонимом типа: int(*)(). Этот тип будет описывать указатель на функцию, которая ничего не принимает и возвращает значение типа: int. Теперь реализуйте конструктор ItemMenu. Который будет принимать указатель на С-строку и указатель на функцию. Реализацию конструктора выполните в файле исходного кода. Также, создайте виртуальный геттер для поля названия пункта. И для него же создайте виртуальную функцию, которая будет печатать этот пункт в консоли. Реализацию виртуальных функций выполните в файле исходного кода. Также, создайте виртуальную функцию run(), которая будет запускать функцию, которая хранится в поле класса. Данная виртуальная функция будет возвращать значение, которое вернула запускаемая функция.

Задача 6

В заголовочном файле, в классе CMenu добавьте в закрытые поля поле, которое будет хранить указатель на наш массив объектов нашего абстрактного класса: ItemMenu, назовите это поле: m_items. По умолчанию нулевой указатель. Теперь реализуйте конструктор для меню, который будет принимать название меню, массив объектов типа элемента меню и их количество. Реализацию также вынесете в файл исходного кода. Также, реализуйте геттеры для всех полей класса.

Задача 7

Последним заданием в этой Л/Р будет создание двух методов класса меню: метода печати всех пунктов на экран (print) и метода считывания пользовательского ввода (runCommand). Последний метод не должен пока обрабатывать корректность этого ввода, данная функциональность будет рассмотрена в другой работе. Также, метод считывания пользовательского ввода, должен возвращать результат выполнения функции, которую он запускает. Полная функциональность его будет следующей: метод запускает функцию вывода на экран всех пунктов и приглашает пользователя сделать ввод. После ввода, он находит в массиве элементов нужный пункт по индексу и запускает метод run(). Возвращаемое значение этого метода, он также возвращает.

Индивидуальная часть

Задача 1

Протестируйте созданное пользовательское меню. Создайте классы наследники, которые будут имплементировать методы абстрактного класса: ItemMenu. Для этого реализуйте функции-«заглушки», которые впоследствии будут выполнять логику вашего задания. Подумайте, какие перегрузки можно сделать для виртуальных функций этого класса? Реализуйте эти перегрузки.

Задача 2

В следующей Л/Р Вы начнёте реализовывать непосредственно свою программу. Для начала подумайте над структурой будущего программного продукта. Разбейте его на смысловые блоки и модули. Создайте эту иерархию представлений в виде папок и классов.

В начало

Вариант №6

Общая задача

Вам будет предложено написать программу – «Автоматизированная система мастерской». Которая будет включать следующий функционал:

- Ведение базы сотрудников и клиентов (разные базы)
  - Создание / удаление / редактирование записей
  - Сортировка / фильтрация
- Ведение базы услуг
- Возможность авторизации
- Создание файлов-отчётов и сохранения состояния

Базовая часть

Задача 1

Создайте новый проект в студии, или создайте новую директорию, в которой будут храниться исходные файлы вашей программы. В этой директории/проекте создайте папку под названием: «MyConMenu». Это будет

ваша рабочая директория, в которой будут лежать исходные файлы библиотеки. В корне создайте файл: «CMenu.cpp» и «CMenu.h».

Задача 2

Реализовывать функционал меню Вы будете в новом пространстве имён, которое будет изолировать ваши имена объектов от сущностей из других библиотек. Для начала создайте новое пространство имён в файле заголовка и в файле исходного кода. Название этого пространства имён будет включать ваши инициалы, пример: ФИО.

Задача 3

В заголовочном файле в новом пространстве имён создайте класс: CMenu, который будет включать следующие закрытые поля: поле типа: int «m_select», которое будет содержать последний ввод пользователя, по умолчанию равняется: -1, поле типа: bool «m_running», которое будет отвечать за выполнение меню, по умолчанию: false, поле типа: char* «m_title», которое будет содержать ссылку на С-строку, которая будет хранить заголовок вашего меню, по умолчанию нулевой указатель и поле типа: size_t «m_count», которое будет хранить количество пунктов меню.

Задача 4

Ваше меню, будет принимать в себя заголовок и массив классов, каждый из которых будет представлять обёртку для запуска нужной функции. Теперь давайте создадим абстрактный класс, от которого впоследствии будем наследовать другие классы с пунктом меню. Для этого в директории «MyConMenu», создайте два новых файла: «AbstractItemMenu.cpp» и «AbstractItemMenu.h». В этих файлах пропишите своё пространство имён. В заголовочном файле, в пространстве имён, создайте класс: «ItemMenu». В этом классе реализуйте следующие закрытые поля: «m_item_name» типа char*, который будет содержать указатель на С-строку названия пункта, по умолчанию – нулевой указатель. И поле: «m_func» типа Func, это поле будет содержать указатель на запускаемую функцию. По умолчанию нулевой указатель.

Задача 5

В классе ItemMenu создайте новую обёртку для прототипа функции через typedef, которая будет называться Func и являться псевдонимом типа: int(*)(). Этот тип будет описывать указатель на функцию, которая ничего не принимает и возвращает значение типа: int. Теперь реализуйте конструктор ItemMenu. Который будет принимать указатель на С-строку и указатель на функцию. Реализацию конструктора выполните в файле исходного кода. Также, создайте виртуальный геттер для поля названия пункта. И для него же создайте виртуальную функцию, которая будет печатать этот пункт в консоли. Реализацию виртуальных функций выполните в файле исходного кода. Также, создайте виртуальную функцию run(), которая будет запускать функцию,

которая хранится в поле класса. Данная виртуальная функция будет возвращать значение, которое вернула запускаемая функция.

Задача 6

В заголовочном файле, в классе CMenu добавьте в закрытые поля поле, которое будет хранить указатель на наш массив объектов нашего абстрактного класса: ItemMenu, назовите это поле: m_items. По умолчанию нулевой указатель. Теперь реализуйте конструктор для меню, который будет принимать название меню, массив объектов типа элемента меню и их количество. Реализацию также вынесете в файл исходного кода. Также, реализуйте геттеры для всех полей класса.

Задача 7

Последним заданием в этой Л/Р будет создание двух методов класса меню: метода печати всех пунктов на экран (print) и метода считывания пользовательского ввода (runCommand). Последний метод не должен пока обрабатывать корректность этого ввода, данная функциональность будет рассмотрена в другой работе. Также, метод считывания пользовательского ввода, должен возвращать результат выполнения функции, которую он запускает. Полная функциональность его будет следующей: метод запускает функцию вывода на экран всех пунктов и приглашает пользователя сделать ввод. После ввода, он находит в массиве элементов нужный пункт по индексу и запускает метод run(). Возвращаемое значение этого метода, он также возвращает.

Индивидуальная часть

Задача 1

Протестируйте созданное пользовательское меню. Создайте классы наследники, которые будут имплементировать методы абстрактного класса: ItemMenu. Для этого реализуйте функции-«заглушки», которые впоследствии будут выполнять логику вашего задания. Подумайте, какие перегрузки можно сделать для виртуальных функций этого класса? Реализуйте эти перегрузки.

Задача 2

В следующей Л/Р Вы начнёте реализовывать непосредственно свою программу. Для начала подумайте над структурой будущего программного продукта. Разбейте его на смысловые блоки и модули. Создайте эту иерархию представлений в виде папок и классов.

В начало

Вариант №7

Общая задача

Вам будет предложено написать программу – «Автоматизированная система кинотеатра». Которая будет включать следующий функционал:

- Ведение базы сотрудников

    - Создание / удаление / редактирование записей
    - Сортировка / фильтрация
- Ведение базы сеансов, фильмов, залов (разные базы)
- Возможность авторизации
- Создание файлов-отчётов и сохранения состояния

Базовая часть

Задача 1

Создайте новый проект в студии, или создайте новую директорию, в которой будут храниться исходные файлы вашей программы. В этой директории/проекте создайте папку под названием: «MyConMenu». Это будет ваша рабочая директория, в которой будут лежать исходные файлы библиотеки. В корне создайте файл: «CMenu.cpp» и «CMenu.h».

Задача 2

Реализовывать функционал меню Вы будете в новом пространстве имён, которое будет изолировать ваши имена объектов от сущностей из других библиотек. Для начала создайте новое пространство имён в файле заголовка и в файле исходного кода. Название этого пространства имён будет включать ваши инициалы, пример: ФИО.

Задача 3

В заголовочном файле в новом пространстве имён создайте класс: CMenu, который будет включать следующие закрытые поля: поле типа: int «m_select», которое будет содержать последний ввод пользователя, по умолчанию равняется: -1, поле типа: bool «m_running», которое будет отвечать за выполнение меню, по умолчанию: false, поле типа: char* «m_title», которое будет содержать ссылку на С-строку, которая будет хранить заголовок вашего меню, по умолчанию нулевой указатель и поле типа: size_t «m_count», которое будет хранить количество пунктов меню.

Задача 4

Ваше меню, будет принимать в себя заголовок и массив классов, каждый из которых будет представлять обёртку для запуска нужной функции. Теперь давайте создадим абстрактный класс, от которого впоследствии будем наследовать другие классы с пунктом меню. Для этого в директории «MyConMenu», создайте два новых файла: «AbstractItemMenu.cpp» и «AbstractItemMenu.h». В этих файлах пропишите своё пространство имён. В заголовочном файле, в пространстве имён, создайте класс: «ItemMenu». В этом классе реализуйте следующие закрытые поля: «m_item_name» типа char*, который будет содержать указатель на С-строку названия пункта, по умолчанию – нулевой указатель. И поле: «m_func» типа Func, это поле будет содержать указатель на запускаемую функцию. По умолчанию нулевой указатель.

Задача 5

В классе ItemMenu создайте новую обёртку для прототипа функции через typedef, которая будет называться Func и являться псевдонимом типа: int(*)(). Этот тип будет описывать указатель на функцию, которая ничего не принимает и возвращает значение типа: int. Теперь реализуйте конструктор ItemMenu. Который будет принимать указатель на С-строку и указатель на функцию. Реализацию конструктора выполните в файле исходного кода. Также, создайте виртуальный геттер для поля названия пункта. И для него же создайте виртуальную функцию, которая будет печатать этот пункт в консоли. Реализацию виртуальных функций выполните в файле исходного кода. Также, создайте виртуальную функцию run(), которая будет запускать функцию, которая хранится в поле класса. Данная виртуальная функция будет возвращать значение, которое вернула запускаемая функция.

Задача 6

В заголовочном файле, в классе CMenu добавьте в закрытые поля поле, которое будет хранить указатель на наш массив объектов нашего абстрактного класса: ItemMenu, назовите это поле: m_items. По умолчанию нулевой указатель. Теперь реализуйте конструктор для меню, который будет принимать название меню, массив объектов типа элемента меню и их количество. Реализацию также вынесете в файл исходного кода. Также, реализуйте геттеры для всех полей класса.

Задача 7

Последним заданием в этой Л/Р будет создание двух методов класса меню: метода печати всех пунктов на экран (print) и метода считывания пользовательского ввода (runCommand). Последний метод не должен пока обрабатывать корректность этого ввода, данная функциональность будет рассмотрена в другой работе. Также, метод считывания пользовательского ввода, должен возвращать результат выполнения функции, которую он запускает. Полная функциональность его будет следующей: метод запускает функцию вывода на экран всех пунктов и приглашает пользователя сделать ввод. После ввода, он находит в массиве элементов нужный пункт по индексу и запускает метод run(). Возвращаемое значение этого метода, он также возвращает.

Индивидуальная часть

Задача 1

Протестируйте созданное пользовательское меню. Создайте классы наследники, которые будут имплементировать методы абстрактного класса: ItemMenu. Для этого реализуйте функции-«заглушки», которые впоследствии будут выполнять логику вашего задания. Подумайте, какие перегрузки можно сделать для виртуальных функций этого класса? Реализуйте эти перегрузки.

Задача 2

В следующей Л/Р Вы начнёте реализовывать непосредственно свою программу. Для начала подумайте над структурой будущего программного продукта. Разбейте его на смысловые блоки и модули. Создайте эту иерархию представлений в виде папок и классов.

В начало

Вариант №8

Общая задача

Вам будет предложено написать программу – «Автоматизированная система диалога (чат бот)». Которая будет включать следующий функционал:

- Ведение базы пользователей
  - Создание / удаление / редактирование записей
  - Сортировка / фильтрация
- Ведение базы диалогов, тем, интересов и напоминаний
- Возможность авторизации
- Создание файлов-отчётов и сохранения состояния

Базовая часть

Задача 1

Создайте новый проект в студии, или создайте новую директорию, в которой будут храниться исходные файлы вашей программы. В этой директории/проекте создайте папку под названием: «MyConMenu». Это будет ваша рабочая директория, в которой будут лежать исходные файлы библиотеки. В корне создайте файл: «CMenu.cpp» и «CMenu.h».

Задача 2

Реализовывать функционал меню Вы будете в новом пространстве имён, которое будет изолировать ваши имена объектов от сущностей из других библиотек. Для начала создайте новое пространство имён в файле заголовка и в файле исходного кода. Название этого пространства имён будет включать ваши инициалы, пример: ФИО.

Задача 3

В заголовочном файле в новом пространстве имён создайте класс: CMenu, который будет включать следующие закрытые поля: поле типа: int «m_select», которое будет содержать последний ввод пользователя, по умолчанию равняется: -1, поле типа: bool «m_running», которое будет отвечать за выполнение меню, по умолчанию: false, поле типа: char* «m_title», которое будет содержать ссылку на С-строку, которая будет хранить заголовок вашего меню, по умолчанию нулевой указатель и поле типа: size_t «m_count», которое будет хранить количество пунктов меню.

Задача 4

Ваше меню, будет принимать в себя заголовок и массив классов, каждый из которых будет представлять обёртку для запуска нужной функции. Теперь давайте создадим абстрактный класс, от которого впоследствии будем наследовать другие классы с пунктом меню. Для этого в директории «MyConMenu», создайте два новых файла: «AbstractItemMenu.cpp» и «AbstractItemMenu.h». В этих файлах пропишите своё пространство имён. В заголовочном файле, в пространстве имён, создайте класс: «ItemMenu». В этом классе реализуйте следующие закрытые поля: «m_item_name» типа char*, который будет содержать указатель на С-строку названия пункта, по умолчанию – нулевой указатель. И поле: «m_func» типа Func, это поле будет содержать указатель на запускаемую функцию. По умолчанию нулевой указатель.

Задача 5

В классе ItemMenu создайте новую обёртку для прототипа функции через typedef, которая будет называться Func и являться псевдонимом типа: int(*)(). Этот тип будет описывать указатель на функцию, которая ничего не принимает и возвращает значение типа: int. Теперь реализуйте конструктор ItemMenu. Который будет принимать указатель на С-строку и указатель на функцию. Реализацию конструктора выполните в файле исходного кода. Также, создайте виртуальный геттер для поля названия пункта. И для него же создайте виртуальную функцию, которая будет печатать этот пункт в консоли. Реализацию виртуальных функций выполните в файле исходного кода. Также, создайте виртуальную функцию run(), которая будет запускать функцию, которая хранится в поле класса. Данная виртуальная функция будет возвращать значение, которое вернула запускаемая функция.

Задача 6

В заголовочном файле, в классе CMenu добавьте в закрытые поля поле, которое будет хранить указатель на наш массив объектов нашего абстрактного класса: ItemMenu, назовите это поле: m_items. По умолчанию нулевой указатель. Теперь реализуйте конструктор для меню, который будет принимать название меню, массив объектов типа элемента меню и их количество. Реализацию также вынесете в файл исходного кода. Также, реализуйте геттеры для всех полей класса.

Задача 7

Последним заданием в этой Л/Р будет создание двух методов класса меню: метода печати всех пунктов на экран (print) и метода считывания пользовательского ввода (runCommand). Последний метод не должен пока обрабатывать корректность этого ввода, данная функциональность будет рассмотрена в другой работе. Также, метод считывания пользовательского ввода, должен возвращать результат выполнения функции, которую он запускает. Полная функциональность его будет следующей: метод запускает

функцию вывода на экран всех пунктов и приглашает пользователя сделать ввод. После ввода, он находит в массиве элементов нужный пункт по индексу и запускает метод run(). Возвращаемое значение этого метода, он также возвращает.

Индивидуальная часть

Задача 1

Протестируйте созданное пользовательское меню. Создайте классы наследники, которые будут имплементировать методы абстрактного класса: ItemMenu. Для этого реализуйте функции-«заглушки», которые впоследствии будут выполнять логику вашего задания. Подумайте, какие перегрузки можно сделать для виртуальных функций этого класса? Реализуйте эти перегрузки.

Задача 2

В следующей Л/Р Вы начнёте реализовывать непосредственно свою программу. Для начала подумайте над структурой будущего программного продукта. Разбейте его на смысловые блоки и модули. Создайте эту иерархию представлений в виде папок и классов.

В начало

Вариант №9

Общая задача

Вам будет предложено написать программу – «Автоматизированная система поликлиники». Которая будет включать следующий функционал:

- Ведение базы врачей и пациентов (разные базы)
  - Создание / удаление / редактирование записей
  - Сортировка / фильтрация
- Ведение базы приёмов и записей в амбулаторных картах
- Возможность авторизации
- Создание файлов-отчётов и сохранения состояния

Базовая часть

Задача 1

Создайте новый проект в студии, или создайте новую директорию, в которой будут храниться исходные файлы вашей программы. В этой директории/проекте создайте папку под названием: «MyConMenu». Это будет ваша рабочая директория, в которой будут лежать исходные файлы библиотеки. В корне создайте файл: «CMenu.cpp» и «CMenu.h».

Задача 2

Реализовывать функционал меню Вы будете в новом пространстве имён, которое будет изолировать ваши имена объектов от сущностей из других библиотек. Для начала создайте новое пространство имён в файле заголовка и

в файле исходного кода. Название этого пространства имён будет включать ваши инициалы, пример: ФИО.

Задача 3

В заголовочном файле в новом пространстве имён создайте класс: CMenu, который будет включать следующие закрытые поля: поле типа: int «m_select», которое будет содержать последний ввод пользователя, по умолчанию равняется: -1, поле типа: bool «m_running», которое будет отвечать за выполнение меню, по умолчанию: false, поле типа: char* «m_title», которое будет содержать ссылку на С-строку, которая будет хранить заголовок вашего меню, по умолчанию нулевой указатель и поле типа: size_t «m_count», которое будет хранить количество пунктов меню.

Задача 4

Ваше меню, будет принимать в себя заголовок и массив классов, каждый из которых будет представлять обёртку для запуска нужной функции. Теперь давайте создадим абстрактный класс, от которого впоследствии будем наследовать другие классы с пунктом меню. Для этого в директории «MyConMenu», создайте два новых файла: «AbstractItemMenu.cpp» и «AbstractItemMenu.h». В этих файлах пропишите своё пространство имён. В заголовочном файле, в пространстве имён, создайте класс: «ItemMenu». В этом классе реализуйте следующие закрытые поля: «m_item_name» типа char*, который будет содержать указатель на С-строку названия пункта, по умолчанию – нулевой указатель. И поле: «m_func» типа Func, это поле будет содержать указатель на запускаемую функцию. По умолчанию нулевой указатель.

Задача 5

В классе ItemMenu создайте новую обёртку для прототипа функции через typedef, которая будет называться Func и являться псевдонимом типа: int(*)(). Этот тип будет описывать указатель на функцию, которая ничего не принимает и возвращает значение типа: int. Теперь реализуйте конструктор ItemMenu. Который будет принимать указатель на С-строку и указатель на функцию. Реализацию конструктора выполните в файле исходного кода. Также, создайте виртуальный геттер для поля названия пункта. И для него же создайте виртуальную функцию, которая будет печатать этот пункт в консоли. Реализацию виртуальных функций выполните в файле исходного кода. Также, создайте виртуальную функцию run(), которая будет запускать функцию, которая хранится в поле класса. Данная виртуальная функция будет возвращать значение, которое вернула запускаемая функция.

Задача 6

В заголовочном файле, в классе CMenu добавьте в закрытые поля поле, которое будет хранить указатель на наш массив объектов нашего абстрактного класса: ItemMenu, назовите это поле: m_items. По умолчанию нулевой

указатель. Теперь реализуйте конструктор для меню, который будет принимать название меню, массив объектов типа элемента меню и их количество. Реализацию также вынесете в файл исходного кода. Также, реализуйте геттеры для всех полей класса.

Задача 7

Последним заданием в этой Л/Р будет создание двух методов класса меню: метода печати всех пунктов на экран (print) и метода считывания пользовательского ввода (runCommand). Последний метод не должен пока обрабатывать корректность этого ввода, данная функциональность будет рассмотрена в другой работе. Также, метод считывания пользовательского ввода, должен возвращать результат выполнения функции, которую он запускает. Полная функциональность его будет следующей: метод запускает функцию вывода на экран всех пунктов и приглашает пользователя сделать ввод. После ввода, он находит в массиве элементов нужный пункт по индексу и запускает метод run(). Возвращаемое значение этого метода, он также возвращает.

Индивидуальная часть

Задача 1

Протестируйте созданное пользовательское меню. Создайте классы наследники, которые будут имплементировать методы абстрактного класса: ItemMenu. Для этого реализуйте функции-«заглушки», которые впоследствии будут выполнять логику вашего задания. Подумайте, какие перегрузки можно сделать для виртуальных функций этого класса? Реализуйте эти перегрузки.

Задача 2

В следующей Л/Р Вы начнёте реализовывать непосредственно свою программу. Для начала подумайте над структурой будущего программного продукта. Разбейте его на смысловые блоки и модули. Создайте эту иерархию представлений в виде папок и классов.

В начало

Вариант №10

Общая задача

Вам будет предложено написать программу – «Автоматизированная система почты». Которая будет включать следующий функционал:

- Ведение базы сотрудников клиентов (разные базы)
  - Создание / удаление / редактирование записей
  - Сортировка / фильтрация
- Ведение базы отправлений
- Возможность авторизации
- Создание файлов-отчётов и сохранения состояния

Базовая часть

Задача 1

Создайте новый проект в студии, или создайте новую директорию, в которой будут храниться исходные файлы вашей программы. В этой директории/проекте создайте папку под названием: «MyConMenu». Это будет ваша рабочая директория, в которой будут лежать исходные файлы библиотеки. В корне создайте файл: «CMenu.cpp» и «CMenu.h».

Задача 2

Реализовывать функционал меню Вы будете в новом пространстве имён, которое будет изолировать ваши имена объектов от сущностей из других библиотек. Для начала создайте новое пространство имён в файле заголовка и в файле исходного кода. Название этого пространства имён будет включать ваши инициалы, пример: ФИО.

Задача 3

В заголовочном файле в новом пространстве имён создайте класс: CMenu, который будет включать следующие закрытые поля: поле типа: int «m_select», которое будет содержать последний ввод пользователя, по умолчанию равняется: -1, поле типа: bool «m_running», которое будет отвечать за выполнение меню, по умолчанию: false, поле типа: char* «m_title», которое будет содержать ссылку на С-строку, которая будет хранить заголовок вашего меню, по умолчанию нулевой указатель и поле типа: size_t «m_count», которое будет хранить количество пунктов меню.

Задача 4

Ваше меню, будет принимать в себя заголовок и массив классов, каждый из которых будет представлять обёртку для запуска нужной функции. Теперь давайте создадим абстрактный класс, от которого впоследствии будем наследовать другие классы с пунктом меню. Для этого в директории «MyConMenu», создайте два новых файла: «AbstractItemMenu.cpp» и «AbstractItemMenu.h». В этих файлах пропишите своё пространство имён. В заголовочном файле, в пространстве имён, создайте класс: «ItemMenu». В этом классе реализуйте следующие закрытые поля: «m_item_name» типа char*, который будет содержать указатель на С-строку названия пункта, по умолчанию – нулевой указатель. И поле: «m_func» типа Func, это поле будет содержать указатель на запускаемую функцию. По умолчанию нулевой указатель.

Задача 5

В классе ItemMenu создайте новую обёртку для прототипа функции через typedef, которая будет называться Func и являться псевдонимом типа: int(*)(). Этот тип будет описывать указатель на функцию, которая ничего не принимает и возвращает значение типа: int. Теперь реализуйте конструктор ItemMenu. Который будет принимать указатель на С-строку и указатель на

функцию. Реализацию конструктора выполните в файле исходного кода. Также, создайте виртуальный геттер для поля названия пункта. И для него же создайте виртуальную функцию, которая будет печатать этот пункт в консоли. Реализацию виртуальных функций выполните в файле исходного кода. Также, создайте виртуальную функцию run(), которая будет запускать функцию, которая хранится в поле класса. Данная виртуальная функция будет возвращать значение, которое вернула запускаемая функция.

Задача 6

В заголовочном файле, в классе CMenu добавьте в закрытые поля поле, которое будет хранить указатель на наш массив объектов нашего абстрактного класса: ItemMenu, назовите это поле: m_items. По умолчанию нулевой указатель. Теперь реализуйте конструктор для меню, который будет принимать название меню, массив объектов типа элемента меню и их количество. Реализацию также вынесете в файл исходного кода. Также, реализуйте геттеры для всех полей класса.

Задача 7

Последним заданием в этой Л/Р будет создание двух методов класса меню: метода печати всех пунктов на экран (print) и метода считывания пользовательского ввода (runCommand). Последний метод не должен пока обрабатывать корректность этого ввода, данная функциональность будет рассмотрена в другой работе. Также, метод считывания пользовательского ввода, должен возвращать результат выполнения функции, которую он запускает. Полная функциональность его будет следующей: метод запускает функцию вывода на экран всех пунктов и приглашает пользователя сделать ввод. После ввода, он находит в массиве элементов нужный пункт по индексу и запускает метод run(). Возвращаемое значение этого метода, он также возвращает.

Индивидуальная часть

Задача 1

Протестируйте созданное пользовательское меню. Создайте классы наследники, которые будут имплементировать методы абстрактного класса: ItemMenu. Для этого реализуйте функции-«заглушки», которые впоследствии будут выполнять логику вашего задания. Подумайте, какие перегрузки можно сделать для виртуальных функций этого класса? Реализуйте эти перегрузки.

Задача 2

В следующей Л/Р Вы начнёте реализовывать непосредственно свою программу. Для начала подумайте над структурой будущего программного продукта. Разбейте его на смысловые блоки и модули. Создайте эту иерархию представлений в виде папок и классов.

В начало

Вариант №11

Общая задача

Вам будет предложено написать программу – «Автоматизированная система гостиницы». Которая будет включать следующий функционал:

- Ведение базы сотрудников и посетителей
  - Создание / удаление / редактирование записей
  - Сортировка / фильтрация
- Ведение базы номеров и услуг (предусмотреть напоминание о выезде)
- Возможность авторизации
- Создание файлов-отчётов и сохранения состояния

Базовая часть

Задача 1

Создайте новый проект в студии, или создайте новую директорию, в которой будут храниться исходные файлы вашей программы. В этой директории/проекте создайте папку под названием: «MyConMenu». Это будет ваша рабочая директория, в которой будут лежать исходные файлы библиотеки. В корне создайте файл: «CMenu.cpp» и «CMenu.h».

Задача 2

Реализовывать функционал меню Вы будете в новом пространстве имён, которое будет изолировать ваши имена объектов от сущностей из других библиотек. Для начала создайте новое пространство имён в файле заголовка и в файле исходного кода. Название этого пространства имён будет включать ваши инициалы, пример: ФИО.

Задача 3

В заголовочном файле в новом пространстве имён создайте класс: CMenu, который будет включать следующие закрытые поля: поле типа: int «m_select», которое будет содержать последний ввод пользователя, по умолчанию равняется: -1, поле типа: bool «m_running», которое будет отвечать за выполнение меню, по умолчанию: false, поле типа: char* «m_title», которое будет содержать ссылку на С-строку, которая будет хранить заголовок вашего меню, по умолчанию нулевой указатель и поле типа: size_t «m_count», которое будет хранить количество пунктов меню.

Задача 4

Ваше меню, будет принимать в себя заголовок и массив классов, каждый из которых будет представлять обёртку для запуска нужной функции. Теперь давайте создадим абстрактный класс, от которого впоследствии будем наследовать другие классы с пунктом меню. Для этого в директории «MyConMenu», создайте два новых файла: «AbstractItemMenu.cpp» и

«AbstractItemMenu.h». В этих файлах пропишите своё пространство имён. В заголовочном файле, в пространстве имён, создайте класс: «ItemMenu». В этом классе реализуйте следующие закрытые поля: «m_item_name» типа char*, который будет содержать указатель на С-строку названия пункта, по умолчанию – нулевой указатель. И поле: «m_func» типа Func, это поле будет содержать указатель на запускаемую функцию. По умолчанию нулевой указатель.

Задача 5

В классе ItemMenu создайте новую обёртку для прототипа функции через typedef, которая будет называться Func и являться псевдонимом типа: int(*)(). Этот тип будет описывать указатель на функцию, которая ничего не принимает и возвращает значение типа: int. Теперь реализуйте конструктор ItemMenu. Который будет принимать указатель на С-строку и указатель на функцию. Реализацию конструктора выполните в файле исходного кода. Также, создайте виртуальный геттер для поля названия пункта. И для него же создайте виртуальную функцию, которая будет печатать этот пункт в консоли. Реализацию виртуальных функций выполните в файле исходного кода. Также, создайте виртуальную функцию run(), которая будет запускать функцию, которая хранится в поле класса. Данная виртуальная функция будет возвращать значение, которое вернула запускаемая функция.

Задача 6

В заголовочном файле, в классе CMenu добавьте в закрытые поля поле, которое будет хранить указатель на наш массив объектов нашего абстрактного класса: ItemMenu, назовите это поле: m_items. По умолчанию нулевой указатель. Теперь реализуйте конструктор для меню, который будет принимать название меню, массив объектов типа элемента меню и их количество. Реализацию также вынесете в файл исходного кода. Также, реализуйте геттеры для всех полей класса.

Задача 7

Последним заданием в этой Л/Р будет создание двух методов класса меню: метода печати всех пунктов на экран (print) и метода считывания пользовательского ввода (runCommand). Последний метод не должен пока обрабатывать корректность этого ввода, данная функциональность будет рассмотрена в другой работе. Также, метод считывания пользовательского ввода, должен возвращать результат выполнения функции, которую он запускает. Полная функциональность его будет следующей: метод запускает функцию вывода на экран всех пунктов и приглашает пользователя сделать ввод. После ввода, он находит в массиве элементов нужный пункт по индексу и запускает метод run(). Возвращаемое значение этого метода, он также возвращает.

Индивидуальная часть

Задача 1

Протестируйте созданное пользовательское меню. Создайте классы наследники, которые будут имплементировать методы абстрактного класса: ItemMenu. Для этого реализуйте функции-«заглушки», которые впоследствии будут выполнять логику вашего задания. Подумайте, какие перегрузки можно сделать для виртуальных функций этого класса? Реализуйте эти перегрузки.

Задача 2

В следующей Л/Р Вы начнёте реализовывать непосредственно свою программу. Для начала подумайте над структурой будущего программного продукта. Разбейте его на смысловые блоки и модули. Создайте эту иерархию представлений в виде папок и классов.

В начало

Вариант №12

Общая задача

Вам будет предложено написать программу – «Автоматизированная система завода». Которая будет включать следующий функционал:

- Ведение базы сотрудников и поставщиков (разные базы)
  - Создание / удаление / редактирование записей
  - Сортировка / фильтрация
- Ведение базы производства и продаж
- Возможность авторизации
- Создание файлов-отчётов и сохранения состояния

Базовая часть

Задача 1

Создайте новый проект в студии, или создайте новую директорию, в которой будут храниться исходные файлы вашей программы. В этой директории/проекте создайте папку под названием: «MyConMenu». Это будет ваша рабочая директория, в которой будут лежать исходные файлы библиотеки. В корне создайте файл: «CMenu.cpp» и «CMenu.h».

Задача 2

Реализовывать функционал меню Вы будете в новом пространстве имён, которое будет изолировать ваши имена объектов от сущностей из других библиотек. Для начала создайте новое пространство имён в файле заголовка и в файле исходного кода. Название этого пространства имён будет включать ваши инициалы, пример: ФИО.

Задача 3

В заголовочном файле в новом пространстве имён создайте класс: CMenu, который будет включать следующие закрытые поля: поле типа: int

«m_select», которое будет содержать последний ввод пользователя, по умолчанию равняется: -1, поле типа: bool «m_running», которое будет отвечать за выполнение меню, по умолчанию: false, поле типа: char* «m_title», которое будет содержать ссылку на С-строку, которая будет хранить заголовок вашего меню, по умолчанию нулевой указатель и поле типа: size_t «m_count», которое будет хранить количество пунктов меню.

Задача 4

Ваше меню, будет принимать в себя заголовок и массив классов, каждый из которых будет представлять обёртку для запуска нужной функции. Теперь давайте создадим абстрактный класс, от которого впоследствии будем наследовать другие классы с пунктом меню. Для этого в директории «MyConMenu», создайте два новых файла: «AbstractItemMenu.cpp» и «AbstractItemMenu.h». В этих файлах пропишите своё пространство имён. В заголовочном файле, в пространстве имён, создайте класс: «ItemMenu». В этом классе реализуйте следующие закрытые поля: «m_item_name» типа char*, который будет содержать указатель на С-строку названия пункта, по умолчанию – нулевой указатель. И поле: «m_func» типа Func, это поле будет содержать указатель на запускаемую функцию. По умолчанию нулевой указатель.

Задача 5

В классе ItemMenu создайте новую обёртку для прототипа функции через typedef, которая будет называться Func и являться псевдонимом типа: int(*)(). Этот тип будет описывать указатель на функцию, которая ничего не принимает и возвращает значение типа: int. Теперь реализуйте конструктор ItemMenu. Который будет принимать указатель на С-строку и указатель на функцию. Реализацию конструктора выполните в файле исходного кода. Также, создайте виртуальный геттер для поля названия пункта. И для него же создайте виртуальную функцию, которая будет печатать этот пункт в консоли. Реализацию виртуальных функций выполните в файле исходного кода. Также, создайте виртуальную функцию run(), которая будет запускать функцию, которая хранится в поле класса. Данная виртуальная функция будет возвращать значение, которое вернула запускаемая функция.

Задача 6

В заголовочном файле, в классе CMenu добавьте в закрытые поля поле, которое будет хранить указатель на наш массив объектов нашего абстрактного класса: ItemMenu, назовите это поле: m_items. По умолчанию нулевой указатель. Теперь реализуйте конструктор для меню, который будет принимать название меню, массив объектов типа элемента меню и их количество. Реализацию также вынесете в файл исходного кода. Также, реализуйте геттеры для всех полей класса.

Задача 7

Последним заданием в этой Л/Р будет создание двух методов класса меню: метода печати всех пунктов на экран (print) и метода считывания пользовательского ввода (runCommand). Последний метод не должен пока обрабатывать корректность этого ввода, данная функциональность будет рассмотрена в другой работе. Также, метод считывания пользовательского ввода, должен возвращать результат выполнения функции, которую он запускает. Полная функциональность его будет следующей: метод запускает функцию вывода на экран всех пунктов и приглашает пользователя сделать ввод. После ввода, он находит в массиве элементов нужный пункт по индексу и запускает метод run(). Возвращаемое значение этого метода, он также возвращает.

Индивидуальная часть

Задача 1

Протестируйте созданное пользовательское меню. Создайте классы наследники, которые будут имплементировать методы абстрактного класса: ItemMenu. Для этого реализуйте функции-«заглушки», которые впоследствии будут выполнять логику вашего задания. Подумайте, какие перегрузки можно сделать для виртуальных функций этого класса? Реализуйте эти перегрузки.

Задача 2

В следующей Л/Р Вы начнёте реализовывать непосредственно свою программу. Для начала подумайте над структурой будущего программного продукта. Разбейте его на смысловые блоки и модули. Создайте эту иерархию представлений в виде папок и классов.

В начало

Вариант №13

Общая задача

Вам будет предложено написать программу – «Автоматизированная система управления музыкальной коллекцией». Которая будет включать следующий функционал:

- Ведение базы исполнителей и пользователей
  - Создание / удаление / редактирование записей
  - Сортировка / фильтрация
- Ведение базы треков, жанров (реализовать подборку)
- Возможность авторизации
- Создание файлов-отчётов и сохранения состояния

Базовая часть

Задача 1

Создайте новый проект в студии, или создайте новую директорию, в которой будут храниться исходные файлы вашей программы. В этой

директории/проекте создайте папку под названием: «MyConMenu». Это будет ваша рабочая директория, в которой будут лежать исходные файлы библиотеки. В корне создайте файл: «CMenu.cpp» и «CMenu.h».

Задача 2

Реализовывать функционал меню Вы будете в новом пространстве имён, которое будет изолировать ваши имена объектов от сущностей из других библиотек. Для начала создайте новое пространство имён в файле заголовка и в файле исходного кода. Название этого пространства имён будет включать ваши инициалы, пример: ФИО.

Задача 3

В заголовочном файле в новом пространстве имён создайте класс: CMenu, который будет включать следующие закрытые поля: поле типа: int «m_select», которое будет содержать последний ввод пользователя, по умолчанию равняется: -1, поле типа: bool «m_running», которое будет отвечать за выполнение меню, по умолчанию: false, поле типа: char* «m_title», которое будет содержать ссылку на С-строку, которая будет хранить заголовок вашего меню, по умолчанию нулевой указатель и поле типа: size_t «m_count», которое будет хранить количество пунктов меню.

Задача 4

Ваше меню, будет принимать в себя заголовок и массив классов, каждый из которых будет представлять обёртку для запуска нужной функции. Теперь давайте создадим абстрактный класс, от которого впоследствии будем наследовать другие классы с пунктом меню. Для этого в директории «MyConMenu», создайте два новых файла: «AbstractItemMenu.cpp» и «AbstractItemMenu.h». В этих файлах пропишите своё пространство имён. В заголовочном файле, в пространстве имён, создайте класс: «ItemMenu». В этом классе реализуйте следующие закрытые поля: «m_item_name» типа char*, который будет содержать указатель на С-строку названия пункта, по умолчанию – нулевой указатель. И поле: «m_func» типа Func, это поле будет содержать указатель на запускаемую функцию. По умолчанию нулевой указатель.

Задача 5

В классе ItemMenu создайте новую обёртку для прототипа функции через typedef, которая будет называться Func и являться псевдонимом типа: int(*)(). Этот тип будет описывать указатель на функцию, которая ничего не принимает и возвращает значение типа: int. Теперь реализуйте конструктор ItemMenu. Который будет принимать указатель на С-строку и указатель на функцию. Реализацию конструктора выполните в файле исходного кода. Также, создайте виртуальный геттер для поля названия пункта. И для него же создайте виртуальную функцию, которая будет печатать этот пункт в консоли. Реализацию виртуальных функций выполните в файле исходного кода. Также,

создайте виртуальную функцию run(), которая будет запускать функцию, которая хранится в поле класса. Данная виртуальная функция будет возвращать значение, которое вернула запускаемая функция.

Задача 6

В заголовочном файле, в классе CMenu добавьте в закрытые поля поле, которое будет хранить указатель на наш массив объектов нашего абстрактного класса: ItemMenu, назовите это поле: m_items. По умолчанию нулевой указатель. Теперь реализуйте конструктор для меню, который будет принимать название меню, массив объектов типа элемента меню и их количество. Реализацию также вынесете в файл исходного кода. Также, реализуйте геттеры для всех полей класса.

Задача 7

Последним заданием в этой Л/Р будет создание двух методов класса меню: метода печати всех пунктов на экран (print) и метода считывания пользовательского ввода (runCommand). Последний метод не должен пока обрабатывать корректность этого ввода, данная функциональность будет рассмотрена в другой работе. Также, метод считывания пользовательского ввода, должен возвращать результат выполнения функции, которую он запускает. Полная функциональность его будет следующей: метод запускает функцию вывода на экран всех пунктов и приглашает пользователя сделать ввод. После ввода, он находит в массиве элементов нужный пункт по индексу и запускает метод run(). Возвращаемое значение этого метода, он также возвращает.

Индивидуальная часть

Задача 1

Протестируйте созданное пользовательское меню. Создайте классы наследники, которые будут имплементировать методы абстрактного класса: ItemMenu. Для этого реализуйте функции-«заглушки», которые впоследствии будут выполнять логику вашего задания. Подумайте, какие перегрузки можно сделать для виртуальных функций этого класса? Реализуйте эти перегрузки.

Задача 2

В следующей Л/Р Вы начнёте реализовывать непосредственно свою программу. Для начала подумайте над структурой будущего программного продукта. Разбейте его на смысловые блоки и модули. Создайте эту иерархию представлений в виде папок и классов.

В начало

Вариант №14

Общая задача

Вам будет предложено написать программу – «Автоматизированная система электронной почты». Которая будет включать следующий функционал:

- Ведение базы администраторов и пользователей
  - Создание / удаление / редактирование записей
  - Сортировка / фильтрация
- Ведение базы отправлений (реализовать возможность отправки и получения писем)
- Возможность авторизации
- Создание файлов-отчётов и сохранения состояния

Базовая часть

Задача 1

Создайте новый проект в студии, или создайте новую директорию, в которой будут храниться исходные файлы вашей программы. В этой директории/проекте создайте папку под названием: «MyConMenu». Это будет ваша рабочая директория, в которой будут лежать исходные файлы библиотеки. В корне создайте файл: «CMenu.cpp» и «CMenu.h».

Задача 2

Реализовывать функционал меню Вы будете в новом пространстве имён, которое будет изолировать ваши имена объектов от сущностей из других библиотек. Для начала создайте новое пространство имён в файле заголовка и в файле исходного кода. Название этого пространства имён будет включать ваши инициалы, пример: ФИО.

Задача 3

В заголовочном файле в новом пространстве имён создайте класс: CMenu, который будет включать следующие закрытые поля: поле типа: int «m_select», которое будет содержать последний ввод пользователя, по умолчанию равняется: -1, поле типа: bool «m_running», которое будет отвечать за выполнение меню, по умолчанию: false, поле типа: char* «m_title», которое будет содержать ссылку на С-строку, которая будет хранить заголовок вашего меню, по умолчанию нулевой указатель и поле типа: size_t «m_count», которое будет хранить количество пунктов меню.

Задача 4

Ваше меню, будет принимать в себя заголовок и массив классов, каждый из которых будет представлять обёртку для запуска нужной функции. Теперь давайте создадим абстрактный класс, от которого впоследствии будем наследовать другие классы с пунктом меню. Для этого в директории «MyConMenu», создайте два новых файла: «AbstractItemMenu.cpp» и

«AbstractItemMenu.h». В этих файлах пропишите своё пространство имён. В заголовочном файле, в пространстве имён, создайте класс: «ItemMenu». В этом классе реализуйте следующие закрытые поля: «m_item_name» типа char*, который будет содержать указатель на С-строку названия пункта, по умолчанию – нулевой указатель. И поле: «m_func» типа Func, это поле будет содержать указатель на запускаемую функцию. По умолчанию нулевой указатель.

Задача 5

В классе ItemMenu создайте новую обёртку для прототипа функции через typedef, которая будет называться Func и являться псевдонимом типа: int(*)(). Этот тип будет описывать указатель на функцию, которая ничего не принимает и возвращает значение типа: int. Теперь реализуйте конструктор ItemMenu. Который будет принимать указатель на С-строку и указатель на функцию. Реализацию конструктора выполните в файле исходного кода. Также, создайте виртуальный геттер для поля названия пункта. И для него же создайте виртуальную функцию, которая будет печатать этот пункт в консоли. Реализацию виртуальных функций выполните в файле исходного кода. Также, создайте виртуальную функцию run(), которая будет запускать функцию, которая хранится в поле класса. Данная виртуальная функция будет возвращать значение, которое вернула запускаемая функция.

Задача 6

В заголовочном файле, в классе CMenu добавьте в закрытые поля поле, которое будет хранить указатель на наш массив объектов нашего абстрактного класса: ItemMenu, назовите это поле: m_items. По умолчанию нулевой указатель. Теперь реализуйте конструктор для меню, который будет принимать название меню, массив объектов типа элемента меню и их количество. Реализацию также вынесете в файл исходного кода. Также, реализуйте геттеры для всех полей класса.

Задача 7

Последним заданием в этой Л/Р будет создание двух методов класса меню: метода печати всех пунктов на экран (print) и метода считывания пользовательского ввода (runCommand). Последний метод не должен пока обрабатывать корректность этого ввода, данная функциональность будет рассмотрена в другой работе. Также, метод считывания пользовательского ввода, должен возвращать результат выполнения функции, которую он запускает. Полная функциональность его будет следующей: метод запускает функцию вывода на экран всех пунктов и приглашает пользователя сделать ввод. После ввода, он находит в массиве элементов нужный пункт по индексу и запускает метод run(). Возвращаемое значение этого метода, он также возвращает.

Индивидуальная часть

Задача 1

Протестируйте созданное пользовательское меню. Создайте классы наследники, которые будут имплементировать методы абстрактного класса: ItemMenu. Для этого реализуйте функции-«заглушки», которые впоследствии будут выполнять логику вашего задания. Подумайте, какие перегрузки можно сделать для виртуальных функций этого класса? Реализуйте эти перегрузки.

Задача 2

В следующей Л/Р Вы начнёте реализовывать непосредственно свою программу. Для начала подумайте над структурой будущего программного продукта. Разбейте его на смысловые блоки и модули. Создайте эту иерархию представлений в виде папок и классов.

В начало

Вариант №15

Общая задача

Вам будет предложено написать программу – «Автоматизированная система управления фотографиями». Которая будет включать следующий функционал:

- Ведение базы фотографий
  - Создание / удаление / редактирование записей
  - Сортировка / фильтрация
- Ведение базы отмеченных людей на фото
- Возможность авторизации
- Создание файлов-отчётов и сохранения состояния

Базовая часть

Задача 1

Создайте новый проект в студии, или создайте новую директорию, в которой будут храниться исходные файлы вашей программы. В этой директории/проекте создайте папку под названием: «MyConMenu». Это будет ваша рабочая директория, в которой будут лежать исходные файлы библиотеки. В корне создайте файл: «CMenu.cpp» и «CMenu.h».

Задача 2

Реализовывать функционал меню Вы будете в новом пространстве имён, которое будет изолировать ваши имена объектов от сущностей из других библиотек. Для начала создайте новое пространство имён в файле заголовка и в файле исходного кода. Название этого пространства имён будет включать ваши инициалы, пример: ФИО.

Задача 3

В заголовочном файле в новом пространстве имён создайте класс: CMenu, который будет включать следующие закрытые поля: поле типа: int «m_select», которое будет содержать последний ввод пользователя, по умолчанию равняется: -1, поле типа: bool «m_running», которое будет отвечать за выполнение меню, по умолчанию: false, поле типа: char* «m_title», которое будет содержать ссылку на С-строку, которая будет хранить заголовок вашего меню, по умолчанию нулевой указатель и поле типа: size_t «m_count», которое будет хранить количество пунктов меню.

Задача 4

Ваше меню, будет принимать в себя заголовок и массив классов, каждый из которых будет представлять обёртку для запуска нужной функции. Теперь давайте создадим абстрактный класс, от которого впоследствии будем наследовать другие классы с пунктом меню. Для этого в директории «MyConMenu», создайте два новых файла: «AbstractItemMenu.cpp» и «AbstractItemMenu.h». В этих файлах пропишите своё пространство имён. В заголовочном файле, в пространстве имён, создайте класс: «ItemMenu». В этом классе реализуйте следующие закрытые поля: «m_item_name» типа char*, который будет содержать указатель на С-строку названия пункта, по умолчанию – нулевой указатель. И поле: «m_func» типа Func, это поле будет содержать указатель на запускаемую функцию. По умолчанию нулевой указатель.

Задача 5

В классе ItemMenu создайте новую обёртку для прототипа функции через typedef, которая будет называться Func и являться псевдонимом типа: int(*)(). Этот тип будет описывать указатель на функцию, которая ничего не принимает и возвращает значение типа: int. Теперь реализуйте конструктор ItemMenu. Который будет принимать указатель на С-строку и указатель на функцию. Реализацию конструктора выполните в файле исходного кода. Также, создайте виртуальный геттер для поля названия пункта. И для него же создайте виртуальную функцию, которая будет печатать этот пункт в консоли. Реализацию виртуальных функций выполните в файле исходного кода. Также, создайте виртуальную функцию run(), которая будет запускать функцию, которая хранится в поле класса. Данная виртуальная функция будет возвращать значение, которое вернула запускаемая функция.

Задача 6

В заголовочном файле, в классе CMenu добавьте в закрытые поля поле, которое будет хранить указатель на наш массив объектов нашего абстрактного класса: ItemMenu, назовите это поле: m_items. По умолчанию нулевой указатель. Теперь реализуйте конструктор для меню, который будет принимать название меню, массив объектов типа элемента меню и их

количество. Реализацию также вынесете в файл исходного кода. Также, реализуйте геттеры для всех полей класса.

Задача 7

Последним заданием в этой Л/Р будет создание двух методов класса меню: метода печати всех пунктов на экран (print) и метода считывания пользовательского ввода (runCommand). Последний метод не должен пока обрабатывать корректность этого ввода, данная функциональность будет рассмотрена в другой работе. Также, метод считывания пользовательского ввода, должен возвращать результат выполнения функции, которую он запускает. Полная функциональность его будет следующей: метод запускает функцию вывода на экран всех пунктов и приглашает пользователя сделать ввод. После ввода, он находит в массиве элементов нужный пункт по индексу и запускает метод run(). Возвращаемое значение этого метода, он также возвращает.

Индивидуальная часть

Задача 1

Протестируйте созданное пользовательское меню. Создайте классы наследники, которые будут имплементировать методы абстрактного класса: ItemMenu. Для этого реализуйте функции-«заглушки», которые впоследствии будут выполнять логику вашего задания. Подумайте, какие перегрузки можно сделать для виртуальных функций этого класса? Реализуйте эти перегрузки.

Задача 2

В следующей Л/Р Вы начнёте реализовывать непосредственно свою программу. Для начала подумайте над структурой будущего программного продукта. Разбейте его на смысловые блоки и модули. Создайте эту иерархию представлений в виде папок и классов.

В начало

Вариант №16

Общая задача

Вам будет предложено написать программу – «Автоматизированная система вокзала». Которая будет включать следующий функционал:

- Ведение базы сотрудников и пассажиров (разные базы)
  - Создание / удаление / редактирование записей
  - Сортировка / фильтрация
- Ведение базы направлений поездов и самих составов (возможность продажи билетов)
- Возможность авторизации
- Создание файлов-отчётов и сохранения состояния

Базовая часть

Задача 1

Создайте новый проект в студии, или создайте новую директорию, в которой будут храниться исходные файлы вашей программы. В этой директории/проекте создайте папку под названием: «MyConMenu». Это будет ваша рабочая директория, в которой будут лежать исходные файлы библиотеки. В корне создайте файл: «CMenu.cpp» и «CMenu.h».

Задача 2

Реализовывать функционал меню Вы будете в новом пространстве имён, которое будет изолировать ваши имена объектов от сущностей из других библиотек. Для начала создайте новое пространство имён в файле заголовка и в файле исходного кода. Название этого пространства имён будет включать ваши инициалы, пример: ФИО.

Задача 3

В заголовочном файле в новом пространстве имён создайте класс: CMenu, который будет включать следующие закрытые поля: поле типа: int «m_select», которое будет содержать последний ввод пользователя, по умолчанию равняется: -1, поле типа: bool «m_running», которое будет отвечать за выполнение меню, по умолчанию: false, поле типа: char* «m_title», которое будет содержать ссылку на С-строку, которая будет хранить заголовок вашего меню, по умолчанию нулевой указатель и поле типа: size_t «m_count», которое будет хранить количество пунктов меню.

Задача 4

Ваше меню, будет принимать в себя заголовок и массив классов, каждый из которых будет представлять обёртку для запуска нужной функции. Теперь давайте создадим абстрактный класс, от которого впоследствии будем наследовать другие классы с пунктом меню. Для этого в директории «MyConMenu», создайте два новых файла: «AbstractItemMenu.cpp» и «AbstractItemMenu.h». В этих файлах пропишите своё пространство имён. В заголовочном файле, в пространстве имён, создайте класс: «ItemMenu». В этом классе реализуйте следующие закрытые поля: «m_item_name» типа char*, который будет содержать указатель на С-строку названия пункта, по умолчанию – нулевой указатель. И поле: «m_func» типа Func, это поле будет содержать указатель на запускаемую функцию. По умолчанию нулевой указатель.

Задача 5

В классе ItemMenu создайте новую обёртку для прототипа функции через typedef, которая будет называться Func и являться псевдонимом типа: int(*)(). Этот тип будет описывать указатель на функцию, которая ничего не принимает и возвращает значение типа: int. Теперь реализуйте конструктор ItemMenu. Который будет принимать указатель на С-строку и указатель на

функцию. Реализацию конструктора выполните в файле исходного кода. Также, создайте виртуальный геттер для поля названия пункта. И для него же создайте виртуальную функцию, которая будет печатать этот пункт в консоли. Реализацию виртуальных функций выполните в файле исходного кода. Также, создайте виртуальную функцию run(), которая будет запускать функцию, которая хранится в поле класса. Данная виртуальная функция будет возвращать значение, которое вернула запускаемая функция.

Задача 6

В заголовочном файле, в классе CMenu добавьте в закрытые поля поле, которое будет хранить указатель на наш массив объектов нашего абстрактного класса: ItemMenu, назовите это поле: m_items. По умолчанию нулевой указатель. Теперь реализуйте конструктор для меню, который будет принимать название меню, массив объектов типа элемента меню и их количество. Реализацию также вынесете в файл исходного кода. Также, реализуйте геттеры для всех полей класса.

Задача 7

Последним заданием в этой Л/Р будет создание двух методов класса меню: метода печати всех пунктов на экран (print) и метода считывания пользовательского ввода (runCommand). Последний метод не должен пока обрабатывать корректность этого ввода, данная функциональность будет рассмотрена в другой работе. Также, метод считывания пользовательского ввода, должен возвращать результат выполнения функции, которую он запускает. Полная функциональность его будет следующей: метод запускает функцию вывода на экран всех пунктов и приглашает пользователя сделать ввод. После ввода, он находит в массиве элементов нужный пункт по индексу и запускает метод run(). Возвращаемое значение этого метода, он также возвращает.

Индивидуальная часть

Задача 1

Протестируйте созданное пользовательское меню. Создайте классы наследники, которые будут имплементировать методы абстрактного класса: ItemMenu. Для этого реализуйте функции-«заглушки», которые впоследствии будут выполнять логику вашего задания. Подумайте, какие перегрузки можно сделать для виртуальных функций этого класса? Реализуйте эти перегрузки.

Задача 2

В следующей Л/Р Вы начнёте реализовывать непосредственно свою программу. Для начала подумайте над структурой будущего программного продукта. Разбейте его на смысловые блоки и модули. Создайте эту иерархию представлений в виде папок и классов.

В начало

Вариант №17

Общая задача

Вам будет предложено написать программу – «Автоматизированная система магазина». Которая будет включать следующий функционал:

- Ведение базы сотрудников и покупателей (разные базы)
  - Создание / удаление / редактирование записей
  - Сортировка / фильтрация
- Ведение базы товаров и продаж (продажа товара)
- Возможность авторизации
- Создание файлов-отчётов и сохранения состояния

Базовая часть

Задача 1

Создайте новый проект в студии, или создайте новую директорию, в которой будут храниться исходные файлы вашей программы. В этой директории/проекте создайте папку под названием: «MyConMenu». Это будет ваша рабочая директория, в которой будут лежать исходные файлы библиотеки. В корне создайте файл: «CMenu.cpp» и «CMenu.h».

Задача 2

Реализовывать функционал меню Вы будете в новом пространстве имён, которое будет изолировать ваши имена объектов от сущностей из других библиотек. Для начала создайте новое пространство имён в файле заголовка и в файле исходного кода. Название этого пространства имён будет включать ваши инициалы, пример: ФИО.

Задача 3

В заголовочном файле в новом пространстве имён создайте класс: CMenu, который будет включать следующие закрытые поля: поле типа: int «m_select», которое будет содержать последний ввод пользователя, по умолчанию равняется: -1, поле типа: bool «m_running», которое будет отвечать за выполнение меню, по умолчанию: false, поле типа: char* «m_title», которое будет содержать ссылку на С-строку, которая будет хранить заголовок вашего меню, по умолчанию нулевой указатель и поле типа: size_t «m_count», которое будет хранить количество пунктов меню.

Задача 4

Ваше меню, будет принимать в себя заголовок и массив классов, каждый из которых будет представлять обёртку для запуска нужной функции. Теперь давайте создадим абстрактный класс, от которого впоследствии будем наследовать другие классы с пунктом меню. Для этого в директории «MyConMenu», создайте два новых файла: «AbstractItemMenu.cpp» и «AbstractItemMenu.h». В этих файлах пропишите своё пространство имён. В

заголовочном файле, в пространстве имён, создайте класс: «ItemMenu». В этом классе реализуйте следующие закрытые поля: «m_item_name» типа char*, который будет содержать указатель на С-строку названия пункта, по умолчанию – нулевой указатель. И поле: «m_func» типа Func, это поле будет содержать указатель на запускаемую функцию. По умолчанию нулевой указатель.

Задача 5

В классе ItemMenu создайте новую обёртку для прототипа функции через typedef, которая будет называться Func и являться псевдонимом типа: int(*)(). Этот тип будет описывать указатель на функцию, которая ничего не принимает и возвращает значение типа: int. Теперь реализуйте конструктор ItemMenu. Который будет принимать указатель на С-строку и указатель на функцию. Реализацию конструктора выполните в файле исходного кода. Также, создайте виртуальный геттер для поля названия пункта. И для него же создайте виртуальную функцию, которая будет печатать этот пункт в консоли. Реализацию виртуальных функций выполните в файле исходного кода. Также, создайте виртуальную функцию run(), которая будет запускать функцию, которая хранится в поле класса. Данная виртуальная функция будет возвращать значение, которое вернула запускаемая функция.

Задача 6

В заголовочном файле, в классе CMenu добавьте в закрытые поля поле, которое будет хранить указатель на наш массив объектов нашего абстрактного класса: ItemMenu, назовите это поле: m_items. По умолчанию нулевой указатель. Теперь реализуйте конструктор для меню, который будет принимать название меню, массив объектов типа элемента меню и их количество. Реализацию также вынесете в файл исходного кода. Также, реализуйте геттеры для всех полей класса.

Задача 7

Последним заданием в этой Л/Р будет создание двух методов класса меню: метода печати всех пунктов на экран (print) и метода считывания пользовательского ввода (runCommand). Последний метод не должен пока обрабатывать корректность этого ввода, данная функциональность будет рассмотрена в другой работе. Также, метод считывания пользовательского ввода, должен возвращать результат выполнения функции, которую он запускает. Полная функциональность его будет следующей: метод запускает функцию вывода на экран всех пунктов и приглашает пользователя сделать ввод. После ввода, он находит в массиве элементов нужный пункт по индексу и запускает метод run(). Возвращаемое значение этого метода, он также возвращает.

Индивидуальная часть

Задача 1

Протестируйте созданное пользовательское меню. Создайте классы наследники, которые будут имплементировать методы абстрактного класса: ItemMenu. Для этого реализуйте функции-«заглушки», которые впоследствии будут выполнять логику вашего задания. Подумайте, какие перегрузки можно сделать для виртуальных функций этого класса? Реализуйте эти перегрузки.

Задача 2

В следующей Л/Р Вы начнёте реализовывать непосредственно свою программу. Для начала подумайте над структурой будущего программного продукта. Разбейте его на смысловые блоки и модули. Создайте эту иерархию представлений в виде папок и классов.

В начало

Вариант №18

Общая задача

Вам будет предложено написать программу – «Автоматизированная система космодрома». Которая будет включать следующий функционал:

- Ведение базы сотрудников и астронавтов
  - Создание / удаление / редактирование записей
  - Сортировка / фильтрация
- Ведение базы космических судов и запланированных запусков (реализовать возможность запуска – виртуально!)
- Возможность авторизации
- Создание файлов-отчётов и сохранения состояния

Базовая часть

Задача 1

Создайте новый проект в студии, или создайте новую директорию, в которой будут храниться исходные файлы вашей программы. В этой директории/проекте создайте папку под названием: «MyConMenu». Это будет ваша рабочая директория, в которой будут лежать исходные файлы библиотеки. В корне создайте файл: «CMenu.cpp» и «CMenu.h».

Задача 2

Реализовывать функционал меню Вы будете в новом пространстве имён, которое будет изолировать ваши имена объектов от сущностей из других библиотек. Для начала создайте новое пространство имён в файле заголовка и в файле исходного кода. Название этого пространства имён будет включать ваши инициалы, пример: ФИО.

Задача 3

В заголовочном файле в новом пространстве имён создайте класс: CMenu, который будет включать следующие закрытые поля: поле типа: int «m_select», которое будет содержать последний ввод пользователя, по умолчанию равняется: -1, поле типа: bool «m_running», которое будет отвечать за выполнение меню, по умолчанию: false, поле типа: char* «m_title», которое будет содержать ссылку на С-строку, которая будет хранить заголовок вашего меню, по умолчанию нулевой указатель и поле типа: size_t «m_count», которое будет хранить количество пунктов меню.

Задача 4

Ваше меню, будет принимать в себя заголовок и массив классов, каждый из которых будет представлять обёртку для запуска нужной функции. Теперь давайте создадим абстрактный класс, от которого впоследствии будем наследовать другие классы с пунктом меню. Для этого в директории «MyConMenu», создайте два новых файла: «AbstractItemMenu.cpp» и «AbstractItemMenu.h». В этих файлах пропишите своё пространство имён. В заголовочном файле, в пространстве имён, создайте класс: «ItemMenu». В этом классе реализуйте следующие закрытые поля: «m_item_name» типа char*, который будет содержать указатель на С-строку названия пункта, по умолчанию – нулевой указатель. И поле: «m_func» типа Func, это поле будет содержать указатель на запускаемую функцию. По умолчанию нулевой указатель.

Задача 5

В классе ItemMenu создайте новую обёртку для прототипа функции через typedef, которая будет называться Func и являться псевдонимом типа: int(*)(). Этот тип будет описывать указатель на функцию, которая ничего не принимает и возвращает значение типа: int. Теперь реализуйте конструктор ItemMenu. Который будет принимать указатель на С-строку и указатель на функцию. Реализацию конструктора выполните в файле исходного кода. Также, создайте виртуальный геттер для поля названия пункта. И для него же создайте виртуальную функцию, которая будет печатать этот пункт в консоли. Реализацию виртуальных функций выполните в файле исходного кода. Также, создайте виртуальную функцию run(), которая будет запускать функцию, которая хранится в поле класса. Данная виртуальная функция будет возвращать значение, которое вернула запускаемая функция.

Задача 6

В заголовочном файле, в классе CMenu добавьте в закрытые поля поле, которое будет хранить указатель на наш массив объектов нашего абстрактного класса: ItemMenu, назовите это поле: m_items. По умолчанию нулевой указатель. Теперь реализуйте конструктор для меню, который будет принимать название меню, массив объектов типа элемента меню и их

количество. Реализацию также вынесете в файл исходного кода. Также, реализуйте геттеры для всех полей класса.

Задача 7

Последним заданием в этой Л/Р будет создание двух методов класса меню: метода печати всех пунктов на экран (print) и метода считывания пользовательского ввода (runCommand). Последний метод не должен пока обрабатывать корректность этого ввода, данная функциональность будет рассмотрена в другой работе. Также, метод считывания пользовательского ввода, должен возвращать результат выполнения функции, которую он запускает. Полная функциональность его будет следующей: метод запускает функцию вывода на экран всех пунктов и приглашает пользователя сделать ввод. После ввода, он находит в массиве элементов нужный пункт по индексу и запускает метод run(). Возвращаемое значение этого метода, он также возвращает.

Индивидуальная часть

Задача 1

Протестируйте созданное пользовательское меню. Создайте классы наследники, которые будут имплементировать методы абстрактного класса: ItemMenu. Для этого реализуйте функции-«заглушки», которые впоследствии будут выполнять логику вашего задания. Подумайте, какие перегрузки можно сделать для виртуальных функций этого класса? Реализуйте эти перегрузки.

Задача 2

В следующей Л/Р Вы начнёте реализовывать непосредственно свою программу. Для начала подумайте над структурой будущего программного продукта. Разбейте его на смысловые блоки и модули. Создайте эту иерархию представлений в виде папок и классов.

В начало

Вариант №19

Общая задача

Вам будет предложено написать программу – «Автоматизированная система библиотеки». Которая будет включать следующий функционал:

- Ведение базы сотрудников и читателей
  - Создание / удаление / редактирование записей
  - Сортировка / фильтрация
- Ведение базы книг, журналов, авторов
- Возможность авторизации
- Создание файлов-отчётов и сохранения состояния

Базовая часть

Задача 1

Создайте новый проект в студии, или создайте новую директорию, в которой будут храниться исходные файлы вашей программы. В этой директории/проекте создайте папку под названием: «MyConMenu». Это будет ваша рабочая директория, в которой будут лежать исходные файлы библиотеки. В корне создайте файл: «CMenu.cpp» и «CMenu.h».

Задача 2

Реализовывать функционал меню Вы будете в новом пространстве имён, которое будет изолировать ваши имена объектов от сущностей из других библиотек. Для начала создайте новое пространство имён в файле заголовка и в файле исходного кода. Название этого пространства имён будет включать ваши инициалы, пример: ФИО.

Задача 3

В заголовочном файле в новом пространстве имён создайте класс: CMenu, который будет включать следующие закрытые поля: поле типа: int «m_select», которое будет содержать последний ввод пользователя, по умолчанию равняется: -1, поле типа: bool «m_running», которое будет отвечать за выполнение меню, по умолчанию: false, поле типа: char* «m_title», которое будет содержать ссылку на С-строку, которая будет хранить заголовок вашего меню, по умолчанию нулевой указатель и поле типа: size_t «m_count», которое будет хранить количество пунктов меню.

Задача 4

Ваше меню, будет принимать в себя заголовок и массив классов, каждый из которых будет представлять обёртку для запуска нужной функции. Теперь давайте создадим абстрактный класс, от которого впоследствии будем наследовать другие классы с пунктом меню. Для этого в директории «MyConMenu», создайте два новых файла: «AbstractItemMenu.cpp» и «AbstractItemMenu.h». В этих файлах пропишите своё пространство имён. В заголовочном файле, в пространстве имён, создайте класс: «ItemMenu». В этом классе реализуйте следующие закрытые поля: «m_item_name» типа char*, который будет содержать указатель на С-строку названия пункта, по умолчанию – нулевой указатель. И поле: «m_func» типа Func, это поле будет содержать указатель на запускаемую функцию. По умолчанию нулевой указатель.

Задача 5

В классе ItemMenu создайте новую обёртку для прототипа функции через typedef, которая будет называться Func и являться псевдонимом типа: int(*)(). Этот тип будет описывать указатель на функцию, которая ничего не принимает и возвращает значение типа: int. Теперь реализуйте конструктор ItemMenu. Который будет принимать указатель на С-строку и указатель на

функцию. Реализацию конструктора выполните в файле исходного кода. Также, создайте виртуальный геттер для поля названия пункта. И для него же создайте виртуальную функцию, которая будет печатать этот пункт в консоли. Реализацию виртуальных функций выполните в файле исходного кода. Также, создайте виртуальную функцию run(), которая будет запускать функцию, которая хранится в поле класса. Данная виртуальная функция будет возвращать значение, которое вернула запускаемая функция.

Задача 6

В заголовочном файле, в классе CMenu добавьте в закрытые поля поле, которое будет хранить указатель на наш массив объектов нашего абстрактного класса: ItemMenu, назовите это поле: m_items. По умолчанию нулевой указатель. Теперь реализуйте конструктор для меню, который будет принимать название меню, массив объектов типа элемента меню и их количество. Реализацию также вынесете в файл исходного кода. Также, реализуйте геттеры для всех полей класса.

Задача 7

Последним заданием в этой Л/Р будет создание двух методов класса меню: метода печати всех пунктов на экран (print) и метода считывания пользовательского ввода (runCommand). Последний метод не должен пока обрабатывать корректность этого ввода, данная функциональность будет рассмотрена в другой работе. Также, метод считывания пользовательского ввода, должен возвращать результат выполнения функции, которую он запускает. Полная функциональность его будет следующей: метод запускает функцию вывода на экран всех пунктов и приглашает пользователя сделать ввод. После ввода, он находит в массиве элементов нужный пункт по индексу и запускает метод run(). Возвращаемое значение этого метода, он также возвращает.

Индивидуальная часть

Задача 1

Протестируйте созданное пользовательское меню. Создайте классы наследники, которые будут имплементировать методы абстрактного класса: ItemMenu. Для этого реализуйте функции-«заглушки», которые впоследствии будут выполнять логику вашего задания. Подумайте, какие перегрузки можно сделать для виртуальных функций этого класса? Реализуйте эти перегрузки.

Задача 2

В следующей Л/Р Вы начнёте реализовывать непосредственно свою программу. Для начала подумайте над структурой будущего программного продукта. Разбейте его на смысловые блоки и модули. Создайте эту иерархию представлений в виде папок и классов.

В начало

Вариант №20

Общая задача

Вам будет предложено написать программу – «Автоматизированная система аэропорта». Которая будет включать следующий функционал:

- Ведение базы сотрудников и пассажиров (разные базы)
  - Создание / удаление / редактирование записей
  - Сортировка / фильтрация
- Ведение базы рейсов, самолётов и продаж билетов
- Возможность авторизации
- Создание файлов-отчётов и сохранения состояния

Базовая часть

Задача 1

Создайте новый проект в студии, или создайте новую директорию, в которой будут храниться исходные файлы вашей программы. В этой директории/проекте создайте папку под названием: «MyConMenu». Это будет ваша рабочая директория, в которой будут лежать исходные файлы библиотеки. В корне создайте файл: «CMenu.cpp» и «CMenu.h».

Задача 2

Реализовывать функционал меню Вы будете в новом пространстве имён, которое будет изолировать ваши имена объектов от сущностей из других библиотек. Для начала создайте новое пространство имён в файле заголовка и в файле исходного кода. Название этого пространства имён будет включать ваши инициалы, пример: ФИО.

Задача 3

В заголовочном файле в новом пространстве имён создайте класс: CMenu, который будет включать следующие закрытые поля: поле типа: int «m_select», которое будет содержать последний ввод пользователя, по умолчанию равняется: -1, поле типа: bool «m_running», которое будет отвечать за выполнение меню, по умолчанию: false, поле типа: char* «m_title», которое будет содержать ссылку на С-строку, которая будет хранить заголовок вашего меню, по умолчанию нулевой указатель и поле типа: size_t «m_count», которое будет хранить количество пунктов меню.

Задача 4

Ваше меню, будет принимать в себя заголовок и массив классов, каждый из которых будет представлять обёртку для запуска нужной функции. Теперь давайте создадим абстрактный класс, от которого впоследствии будем наследовать другие классы с пунктом меню. Для этого в директории «MyConMenu», создайте два новых файла: «AbstractItemMenu.cpp» и «AbstractItemMenu.h». В этих файлах пропишите своё пространство имён. В

заголовочном файле, в пространстве имён, создайте класс: «ItemMenu». В этом классе реализуйте следующие закрытые поля: «m_item_name» типа char*, который будет содержать указатель на С-строку названия пункта, по умолчанию – нулевой указатель. И поле: «m_func» типа Func, это поле будет содержать указатель на запускаемую функцию. По умолчанию нулевой указатель.

Задача 5

В классе ItemMenu создайте новую обёртку для прототипа функции через typedef, которая будет называться Func и являться псевдонимом типа: int(*)(). Этот тип будет описывать указатель на функцию, которая ничего не принимает и возвращает значение типа: int. Теперь реализуйте конструктор ItemMenu. Который будет принимать указатель на С-строку и указатель на функцию. Реализацию конструктора выполните в файле исходного кода. Также, создайте виртуальный геттер для поля названия пункта. И для него же создайте виртуальную функцию, которая будет печатать этот пункт в консоли. Реализацию виртуальных функций выполните в файле исходного кода. Также, создайте виртуальную функцию run(), которая будет запускать функцию, которая хранится в поле класса. Данная виртуальная функция будет возвращать значение, которое вернула запускаемая функция.

Задача 6

В заголовочном файле, в классе CMenu добавьте в закрытые поля поле, которое будет хранить указатель на наш массив объектов нашего абстрактного класса: ItemMenu, назовите это поле: m_items. По умолчанию нулевой указатель. Теперь реализуйте конструктор для меню, который будет принимать название меню, массив объектов типа элемента меню и их количество. Реализацию также вынесете в файл исходного кода. Также, реализуйте геттеры для всех полей класса.

Задача 7

Последним заданием в этой Л/Р будет создание двух методов класса меню: метода печати всех пунктов на экран (print) и метода считывания пользовательского ввода (runCommand). Последний метод не должен пока обрабатывать корректность этого ввода, данная функциональность будет рассмотрена в другой работе. Также, метод считывания пользовательского ввода, должен возвращать результат выполнения функции, которую он запускает. Полная функциональность его будет следующей: метод запускает функцию вывода на экран всех пунктов и приглашает пользователя сделать ввод. После ввода, он находит в массиве элементов нужный пункт по индексу и запускает метод run(). Возвращаемое значение этого метода, он также возвращает.

Индивидуальная часть

Задача 1

Протестируйте созданное пользовательское меню. Создайте классы наследники, которые будут имплементировать методы абстрактного класса: ItemMenu. Для этого реализуйте функции-«заглушки», которые впоследствии будут выполнять логику вашего задания. Подумайте, какие перегрузки можно сделать для виртуальных функций этого класса? Реализуйте эти перегрузки.

Задача 2

В следующей Л/Р Вы начнёте реализовывать непосредственно свою программу. Для начала подумайте над структурой будущего программного продукта. Разбейте его на смысловые блоки и модули. Создайте эту иерархию представлений в виде папок и классов.

В начало

Контрольные вопросы

1. Что такое иерархия сущностей?
2. Приведите пример иерархии сущностей из реальной жизни?
3. Как воспроизводится иерархия классов?
4. Что такое наследование?
5. Как выполняется наследование класса?
6. Виды наследования, модификаторы наследования.
7. Связь модификаторов доступа и модификаторов наследования.
8. Назовите виды инициализации объектов класса.
9. Что такое виртуальная функция?
10. Что такое чистая виртуальная функция и как её создать?
11. Что такое абстрактный класс?
12. Чем абстрактный класс отличается от обычного?
13. Как создать абстрактный класс?
14. Что такое интерфейс?
15. Как использовать интерфейс?
16. Как переопределять методы класса?
17. Как устроено переопределение виртуальных функций?

Список литературы

1. Курс лекций доцента кафедры ФН1-КФ Пчелинцевой Н.И.
2. Программирование на языке высокого уровня C/C++ [Электронный ресурс]: конспект лекций / – Электрон. текстовые данные. – М.: Московский государственный строительный университет, Ай Пи Эр Медиа, ЭБС АСВ, 2016. – 140 с. – Режим доступа: http://www.iprbookshop.ru/48037.

В начало