Лабораторная работа № 3
Конструирование класса, моделирующего работу устройства

Цель работы:познакомиться с понятием инкапсуляция в объектно-ориентированном программировании, научиться защищать свойства объекта, моделировать поведение предмета.

Задание

1. Разработать модель работы устройства, состоящую из 3-4 свойства, наиболее существенных для описания поведения устройства, и 3-6 методов, моделирующих поведение объекта.
2. Определить набор допустимых значений для каждого свойства.
3. Выявить все допустимые состояния объекта данного класса и представить их в виде таблицы следующего вида:

№ состояния	Свойство 1	Свойство 2
1	Значение 1	Значение 2
2	Значение 3	Значение 4

4. На основе модели сконструировать класс. Доступ к свойствам класса должен быть закрытым (private) , к методам – открытым (public); изменение значений свойств (состояния) объекта осуществляется соответствующими методами.
5. Разработать консольное приложение, демонстрирующее работу объекта, в составе:
   - главный модуль, содержащий функцию main,
   - модуль, содержащий разработанный класс.
6. Программа должна выполнять следующие действия:
   - создание объекта;
   - демонстрация текущего состояния объекта с помощью сообщений на экране;
   - действия над объектом, меняющие его состояние.
7. Выбор действия над объектом осуществить через меню в цикле, чтобы предоставить пользователю возможность выбирать любую последовательность действий.
8. Разработать тесты для проверки соответствия модели поведения устройства и результатов работы программы.
9. Представить отчёт следующего содержания:
   - постановка задачи;
   - диаграмма файлов проекта, содержащих исходный код (UML-диаграмма компонентов);
   - описание класса;
   - текст программы;
   - таблица допустимых состояний.

Варианты заданий:

№	Объект
1	Дорожный светофор
2	Легковой автомобиль
3	Лифт
4	Башенный кран
5	Электронный секундомер
6	Радиоприемник
7	Автоматически регулируемый железнодорожный переезд
8	Цифровой вольтметр
9	Цифровой фотоаппарат
10	Кондиционер
11	Электрический чайник

12	Электрообогреватель
13	Автомат по продаже кофе
14	Сотовый телефон
15	Телевизор
16	Монитор
17	Калькулятор
18	Банкомат
19	Магнитофон
20	Плеер
21	Автоматическая стиральная машина
22	Телефон-факс
23	Накопитель DVD-RW
24	Поезд метро
25	Модем
26	Источник бесперебойного питания
27	Клавиатура
28	Лазерный принтер
29	Цветной струйный принтер
30	Холодильник
31	Тестер
32	Ноутбук
33	Электронная кофеварка
34	Электронный дверной замок
35	СВЧ-печка

Справочный материал.

Конструирование кода класса

Класс рекомендуется помещать в отдельный модуль, состоящий из двух частей: интерфейса модуля и реализации модуля, имеющих одно имя, но разные расширения. В C++ интерфейсная часть модуля класса помещается в заголовочный файл (*.h), содержит объявление класса , директивы препроцессора. В заголовочном файле обязательно должна быть комбинация из набора директив:

```
#ifndef MACROS_NAME
#define MACROS_NAME
 class MyClass
{
public:

private:

};
#endif   //  MACROS_NAME
```

Эта комбинация не позволяет данный файл многократно включать в проект. Реализация модуля помещается в файл *.cpp, *.cc и содержит директивы препроцессора и реализацию функций-членов класса. Для объединения интерфейсной части модуля и его реализации обязательно следует использовать в файле реализации директиву #include, подключающую интерфейс модуля. Т.к. подключаемый заголовочный файл находится в каталоге проекта или в одном из его подкаталогов, имя файла следует записывать в двойных кавычках и правильно указывать к нему

путь с учетом его местонахождения. Путь начинается в каталоге того файла, в который помещается директива #include.

Этапы конструирования проекта

1. Продумать модель заданного устройства и зафиксировать ее на листе бумаги.
2. Создать новый проект на языке C++, при генерации проекта предусмотреть автоматическое добавление функции main(). Заголовок функции main(), полученный автоматически, не редактировать!
3. Добавить модуль класса вручную или с помощью специальной команды инструментальной среды (Add class, Insert class, New class в зависимости от инструментальной среды). При автоматическом добавлении модуля класса с именем MyClass в проект будут включены два файла MyClass.h и MyClass.cpp, в которых уже будет размещена служебная часть кода класса.
4. Ввести код класса, соответствующий разработанной модели.
5. Подключить модуль класса к главному модулю директивой #include "MyClass.h", при этом учесть путь от главного модуля до файла MyClass.h.
6. В функции main() создать объект и ввести код программы управления объектом. Т. к. действия над объектом выполняются в цикле, рекомендуется использовать комбинацию while и switch. В примере используются вспомогательные переменные: work, отвечающая за работу цикла, и command, хранящая номер выбранной команды.

```
int main()
{
	int work = 1, command;
	while (work)
	{
		std::cout <<"Вывод меню" <<std::endl;
		// содержание меню
		std::cout <<"Введите номер команды->";
		std::cin >>command;
		switch(command)
		{
		case 1: // команда 1
				break;
		case 2: // команда 2 и т. д.
				break;
		// . . . .
		case 10: work=0;
		}
	}
	return 0;
}
```

Диаграмма компонентов языка UML

UML (Unified Modelling Language) — это специальный графический язык, предназначенный для моделирования объектно-ориентированных программ и систем. Главные элементы UML — диаграммы, представляющие программу или систему с разных точек зрения. Существуют статические и динамические диаграммы. Статическая диаграмма представляет программу вне зависимости от времени. Динамические диаграммы описывают поведение системы во времени.

Диаграмма компонентов — один из видов диаграмм UML. Это статическая диаграмма, демонстрирующая файловый состав проекта (файлы с исходным кодом, двоичные файлы, динамически загружаемые библиотеки, файлы с данными, и.т.д. и их связи). Для обозначения компонента на диаграмме используется прямоугольник с именем файла или компонента внутри. Для обозначения связей компонентов используются различные типы линий со стрелками и без. Один из видов связи — «использование», отражает зависимость одного компонента от другого. Например, при сборке проекта из исходного кода эта зависимость проявляется при подключении модуля класса к главному модулю (директива #include). Связь типа использование обозначается пунктирной линией со стрелкой в направлении от зависимого файла к независимому (Рисунок 1).

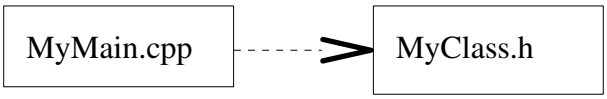

Рисунок 1. Диаграмма компонентов: вид связи «использование».

Вид связи, устанавливаемый для интерфейсной (внешней) части модуля и его реализации, называется «реализация» и обозначается пунктирной линией со стрелкой в виде не закрашенного треугольника, указывающей в сторону интерфейса модуля (Рисунок 2).

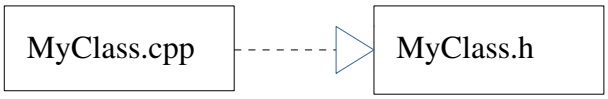

Рисунок 2. Диаграмма компонентов: вид связи «реализация».

Вопросы к защите

1. Чем определяется состояние объекта?
2. Что такое инкапсуляция?
3. Что представляет собой модуль в C/C++?
4. Какая часть кода класса может находиться в файле *.h?
5. В каком файле находится реализация функций-членов класса?
6. Сколько файлов будет содержать проект-консольное приложение, если в проекте созданы два класса?