# Лабораторная работа №2. Создание и модификация классов

**1 Цель работы**

Целью работы – ознакомиться с понятием класса и способами его объявления, изучить понятия конструктора, деструктора, доступности компонентов класса, а также методы работы с компонентными данными и компонентными функциями класса.

**2 Порядок выполнения работы**

* ознакомиться с описанием лабораторной работы;
* получить задание у преподавателя по вариантам;
* разработать и отладить программу;
* составить и защитить отчет по лабораторной работе у преподавателя.

**3 Содержание отчета**

* титульный лист;
* краткое теоретическое описание;
* задание на лабораторную работу, включающее математическую формулировку задачи;
* результаты выполнения работы, включающие схему алгоритма, тексты программ, результаты вычислений;

**4 Краткая теория**

**4.1 Объявление класса**

*Класс* – это тип данных, определяемый пользователем. Понятия класса, структуры и объединения в С++ довольно близки друг к другу. Поэтому почти все, что будет далее говориться о классах, применимо также к структурам и объединениям.

Механизм классов позволяет создавать типы в полном соответствии с принципами абстракции данных, т.е. *класс задает некоторую структурированную совокупность типизированных данных и позволяет определить набор операций над этими данными*.

Класс должен быть объявлен до того, как будет объявлена хотя бы одна переменная этого класса. Т.е. класс не может объявляться внутри объявления переменной.

Синтаксис объявления класса следующий:

**class имя класса**

**{**

**public:**

**<данные, функции>**

**protected:**

**<данные, функции>**

**private:**

**<данные, функции>**

**};**

Например:

**class MyClass**

**{**

**public:**

**MyClass(int=0);**

**void SetA(int);**

**int GetA();**

**private:**

**int FA;**

**double B,C;**

**protected:**

**int F(int);**

**};**

Имя класса может быть любым допустимым идентификатором.

Доступ к объявляемым компонентам класса определяется тем, в каком разделе они объявлены. Раздел **public** (открытый) предназначен для объявлений, которые доступны для внешнего использования. Это открытый интерфейс класса. Раздел **private** (закрытый) содержит объявления полей и функций, используемых только внутри данного класса. Раздел **protected** (защищенный) содержит объявления, доступные только для потомков объявляемого класса.

В приведенном выше примере через объект данного класса можно получить доступ только к функциям **MyClass**, **SetA** и **GetA**. Поля **FA**, **B**, **C** и функция **F** – закрытые элементы. Это вспомогательные данные и функция, которые используют в своей работе открытые функции. Открытая функция **MyClass** с именем, совпадающим с именем класса, это так называемый конструктор класса, который должен инициализировать данные в момент создания объекта класса.

По умолчанию в классах (в отличие от структур) предполагается спецификатор **private**. Поэтому можно включать в объявление класса данные и функции, не указывая спецификатора доступа. Все, что включено в описание до первого спецификатора доступа, считается закрытым.

Объявления компонентных данных (полей) выглядят так же, как объявления переменных или объявления полей в структурах:

**тип имена\_полей;**

В объявлении класса поля запрещается инициализировать. Для инициализации данных служат конструкторы.

Объявления компонентных функций в простейшем случае не отличаются от обычных объявлений функций. Компонентные функции классов также называют *методами*.

После того, как объявлен класс, можно создавать объекты этого класса. Объект класса создается как любая переменная другого типа простым объявлением. Например, оператор

**MyClass MC, MC10[10], \*Pmc;**

создает объект **MC** объявленного выше класса **MyClass**, массив **MC10** из десяти объектов данного класса и указатель **Pmc** на объект этого класса.

В момент создания объекта класса, имеющего конструктор, можно инициализировать его данные, перечисляя в скобках после имени объекта значения данных. Например, оператор

**MyClass MC(3);**

не только создает объект **MC**, но и задает его полю **FA** значение 3. Если этого не сделать, то в момент создания объекта поле получит значение по умолчанию, указанное в содержащемся в объявлении класса прототипе конструктора.

Создание переменных класса, использующих класс, можно совместить с объявлением самого класса, размещая их список между закрывающей класс фигурной скобкой и завершающей точкой с запятой:

**class MyClass**

**{**

**...**

**} MC, MC[10], \*Pmc;**

Если создается динамически размещаемый объект класса, то это делается операцией **new**. Например:

**MyClass \*PMC = new MyClass(3);**

Этот оператор создает в динамической распределяемой области памяти сам объект и создает указатель на него – переменную **PMC**.

После определения объекта класса можно обращаться к его доступным компонентам с помощью “квалифицированных имен”:

**имя\_объекта.имя\_класса::имя\_компонента**

Имя\_класса с “**::**” (операцией уточнения области действия) может быть опущено:

**имя\_объекта.имя\_элемента**

Уточненное имя принадлежащей классу компонентной функции

**имя\_объекта.обращение\_к\_компонентной\_функции**

обеспечивает вызов компонентной функции класса для обработки данных именно того объекта, имя которого использовалось в уточненном имени. Например:

**MC.SetA(5);**

**int n = MC10[4].GetA();**

Другой доступ к элементам объекта некоторого класса предусматривает явное использование указателя на объект класса и операции косвенного выбора компоненты (**->** ):

**указатель\_на\_объект\_класса->имя\_элемента**

Через указатель можно вызвать компонентную функцию:

**указатель\_на\_объект\_класса->вызов\_компонентной\_функции**

Например:

**PMC->SetA(12);**

**4.2 Компонентные функции, дружественные функции, константные функции**

Поля данных, исходя из принципа инкапсуляции, всегда должны быть защищены от несанкционированного доступа. Доступ к ним, как правило, должен осуществляться только через функции, включающие методы чтения и записи полей. В этих функциях должна осуществляться проверка данных, чтобы не записать случайно в поля неверные данные или чтобы не допустить их неверной трактовки.

Поэтому данные всегда целесообразно объявлять в разделе **private** – закрытом разделе класса. В редких случаях их можно помещать в **protected** – защищенном разделе класса, чтобы возможные потомки данного класса имели к ним доступ.

Приведем пример. Реализация функций записи и чтения для ранее объявленного класса **MyClass** может иметь вид:

**void MyClass::SetA(int Value)**

**{**

**if (...) //проверка корректности данных**

**FA = Value;**

**}**

**int MyClass::GetA()**

**{**

**return FA;**

**}**

В данном случае функция чтения просто возвращает значение поля, но в более сложных классах может потребоваться какая-то предварительная обработка данных. Обратите внимание, что все описания функций-элементов содержат ссылку на класс с помощью операции разрешения области видимости (**::**).

В приведенном примере объявление класса содержит только прототипы функций, а их реализация вынесена из описания класса. Для простых функций реализация может быть размещена непосредственно в объявлении класса. Например:

**class MyClass**

**{**

**public:**

**MyClass(int = 0);**

**void SetA(int Value) {FA = Value}; //функция записи**

**int GetA() {return FA}; //функция чтения**

**private:**

**int FA;**

**double B,C;**

**};**

Функции, описание которых содержится в объявлении класса, в действительности являются встраиваемыми функциями **inline**. Введение описания функций в объявление класса – это плохой стиль программирования: следует избегать смешения открытого интерфейса класса, содержащегося в его объявлении, и реализации класса.

Компонентные функции класса имеют доступ к любым другим функциям-элементам и к любым данным-элементам, как открытым, так и закрытым. Функции, не принадлежащие классу, но работающие с его объектами, имеют доступ только к открытым функциям-элементам и данным-элементам. Но в некоторых случаях желательно обеспечить доступ к закрытым компонентам для функций, не являющихся элементами данного класса. Это можно сделать, объявив соответствующую функцию как *дружественную* с помощью спецификатора **friend**. Например, если в объявление класса включить оператор

**friend void IncA(MyClass);**

то функция **IncA**, не являясь элементом данного класса, получает доступ к его закрытым элементам. Например, функция **IncA** может быть описана где-то в программе следующим образом:

**void IncA(MyClass Ob)**

**{**

**Ob.FA++;**

**}**

Дружественными могут быть не только функции, но и целые классы. Например, можно поместить в объявление своего класса оператор

**friend Class1;**

и все компонентные функции класса **Class1** получат доступ к закрытым компонентам описываемого класса.

Иногда программист может захотеть создать объект класса как константный с помощью спецификатора **const**. Например:

**const Class1 MC1(3);**

Если при этом класс содержит не только функции чтения, но и записи данных, то реакция на такой оператор, введенный пользователем, зависит от версии и настройки компилятора. Компилятор может выдать сообщение об ошибке и отказаться от компиляции, а может просто выдать предупреждение и проигнорировать спецификатор пользователя **const**.

Функция класса может быть объявлена так, что ей разрешается только *читать* значения компонентных данных этого класса, но *не изменять* их. Подобные функции должны быть объявлены с суффиксом **const** (т.е. между именем функции и ее телом записывается ключевое слово **const**). Пример:

**int GetA() const;**

**...**

**int MyClass::GetA() const**

**{**

**return FA;**

**}**

**4.3 Компонентные данные, статические данные, константные данные**

Теперь рассмотрим несколько подробнее компонентные данные. Обычно каждый объект класса имеет свою собственную копию всех данных-элементов класса. Но в определенных случаях во всех объектах класса должна фигурировать только одна копия некоторых данных. Например, это может быть счетчик числа созданных объектов класса. Или же, если в классе имеются некоторые константы, одинаковые для всех объектов класса, то нерационально хранить в каждом объекте собственные копии этих констант. Рациональнее иметь единственные экземпляры этих констант для всех объектов.

Для введения в класс подобных данных используются *статические данные*, которые содержат информацию «для всего класса». Объявление статических данных начинается с ключевого слова **static**. Например:

**static int D;**

Статические компоненты могут быть открытыми, закрытыми или защищенными. Доступ к открытым статическим элементам класса возможен посредством любого объекта класса или посредством имени класса с помощью бинарной операции разрешения области действия. Например:

**MyClass::D = 10;**

Закрытые и защищенные статические элементы класса должны быть доступны открытым функциям-элементам этого класса или друзьям класса.

Статические компоненты класса существуют даже тогда, когда не существует никаких объектов этого класса. В этом случае доступ к открытому статическому элементу обеспечивается так же, как указано выше: с помощью имени класса и бинарной операции разрешения области действия. Для обеспечения доступа в отсутствие объектов к закрытому или защищенному компоненту класса должна быть предусмотрена открытая статическая функция-элемент, которая должна вызываться с добавлением перед ее именем имени класса и бинарной операции разрешения области действия.

Начальные значения статических компонентов (как открытых, так и закрытых) должны задаваться вне объявления класса. Для этого достаточно разместить где-то в файле, например, после объявления класса или среди реализаций компонентных функций (но не внутри их) оператор вида

**int MyClass::D = 0;**

Среди компонентных данных могут быть объявлены именованные константы (не во всех компиляторах!). Например:

**static const int MaxA = 10;**

**const int MinA;**

Значения статических именованных констант могут задаваться в момент их объявления в классе, как показано в предыдущем примере. Инициализация нестатических констант будет рассмотрена в следующем разделе.

**4.4 Конструкторы и деструкторы**

*Конструктором класса* называется открытая компонентная функция, которая вызывается в момент создания объекта класса и должна инициализировать данные указанными в вызове значениями или значениями по умолчанию. Конструктор имеет то же имя, что и класс.

Наличие конструктора в классе не обязательно. Но если конструктор отсутствует, то клиенты класса (внешние функции, использующие класс) должны сами заботиться об инициализации данных, т.е. о задании им некоторых начальных значений. Это не всегда возможно. Например, если класс имеет закрытые данные, предназначенные только для чтения, то для этих данных не предусматриваются открытые функции записи. И клиент не в состоянии присвоить данным какие-то начальные значения.

Пример объявления и реализации конструктора:

**class MyClass**

**{**

**public:**

**MyClass(); //конструктор класса**

**...**

**private:**

**int A;**

**...**

**};**

**...**

**MyClass::MyClass()**

**{**

**A = 0;**

**}**

В этом примере объявлен конструктор **MyClass** без параметров, который при создании объекта задает начальное значение поля **А** равным **0**. Следует обратить внимание на то, что в отличие от других функций в объявлении конструктора не указывается тип возвращаемого значения.

Простое задание в конструкторе значений данных в общем случае не гарантирует их целостность. Обычно нужна еще проверка допустимости данных. Например, если в классе есть функция записи **SetA**, осуществляющая такие проверки, то лучше обратиться к ней и при задании начального значения. В этом случае реализация конструктора примет вид:

**MyClass::MyClass()**

**{**

**SetA(0);**

**}**

Создание объекта описанного класса **MyClass** в программе должно осуществляться или объявлением соответствующей переменной или динамическим размещением переменной в памяти. В момент выполнения каждого из этих операторов неявным образом вызывается конструктор, устанавливающий начальные значения данных.

Недостатком конструкторов показанного типа является то, что все начальные значения данных задаются в них конструктором. Вызывающая функция никак не может вмешаться в этот процесс и задать какое-то другое значение.

Другой крайностью являются конструкторы, в которых все начальные значения задаются как параметры. Например, прототип конструктора может иметь вид:

**MyClass(int);**

а его реализация:

**MyClass::MyClass(int a)**

**{**

**SetA(a);**

**}**

В этом случае поле **FA** инициализируется параметром, передаваемым в конструктор. Такой конструктор обычно тоже неудобен, поскольку в классе может быть много параметров и задавать значения их всех при создании объекта очень громоздко и чревато ошибками.

Чаще всего используются конструкторы с параметрами по умолчанию. В этом случае объявление конструктора может иметь вид:

**MyClass(int = 0);**

а его реализация:

**MyClass::MyClass(int a)**

**{**

**SetA(a);**

**}**

Объект такого класса можно задавать любым из приведенных ранее операторов создания объекта. Если при создании указывается аргумент, то его значение присваивается полю. Если аргумент не указывается, то присваивается значение по умолчанию (в нашем примере 0). Этот вариант конструктора наиболее гибкий. Поэтому он чаще всего используется при построении классов.

В объявлении класса могут быть определены не только поля переменных, но и некоторые именованные константы. Например,

**const int MaxA;**

Если такая константа объявлена как статическая, то в ее объявление в классе можно непосредственно включить инициализацию (не во всех версиях С++!):

**static const int MaxA = 10;**

Но тогда это значение нельзя изменить. А задать значение такой константы в конструкторе невозможно, поскольку компилятор не разрешает присваивать значения константам. Выходом из положения является специальный синтаксис конструктора с инициализатором элементов. Инициализатор элементов записывается после заголовка конструктора в его реализации, предваряется двоеточием и содержит имена константных данных, после которых в скобках указываются их значения. Например, если в объявлении класса **MyClass** имеются строки

**const int MaxA;**

**const int MinA;**

то реализацию конструктора такого класса с уже описанным прототипом

**MyClass(int = 0);**

надо дополнить инициализатором элементов:

**MyClass::MyClass(int a) : MaxA(10),MinA(1)**

**{**

**SetA(a);**

**}**

В данном случае инициализатор задает константе **MaxA** начальное значение **10**, а константе **MinA** – значение **1**.

Можно предоставить пользователю возможность изменять значения констант в момент создания объекта. В этом случае в конструкторе надо предусмотреть для констант соответствующие значения по умолчанию:

**MyClass(int A = 0,int MaxA = 10,int MinA = 1);**

или

**MyClass(int = 0,int = 10,int = 1);**

(второй вариант менее удобен, так как не позволяет по прототипу функции понять, в какой последовательности должны задаваться параметры).

Тогда реализацию конструктора можно оформить так:

**MyClass::MyClass(int a,int i,int j) : MaxA(i),MinA(j)**

**{**

**SetA(a);**

**}**

Создание объектов такого типа может осуществляться, например, такими операторами:

**MyClass MC; //умолчание: А= 0, MaxA=10, MinA=1**

**MyClass MC(20); //задано: А=20, MaxA=10, MinA=1**

**MyClass MC(20,15); //задано: А=20, MaxA=15, MinA=1**

**MyClass MC(20,15,2); //задано: А=20, MaxA=15, MinA=2**

Теперь остановимся на *деструкторах*. Это специальные компонентные функции, срабатывающие при уничтожении динамически размещенного объекта класса и освобождающие занимаемую им память. Имя деструктора совпадает с именем класса, но перед ним записывается символ тильда (**~**). Как и для конструктора, в деструкторе не указывается возвращаемый тип. Например:

**class MyClass**

**{**

**public:**

**~MyClass(); //деструктор класса**

**};**

Деструкторы необходимы, если конструктор или какие-то функции-элементы класса динамически распределяют память, создавая в ней какие-то объекты. Тогда деструктор должен эти объекты удалять. В остальных случаях можно обычно обойтись без деструктора.

Если деструктор явным образом не объявлен, компилятор сам генерирует необходимые коды освобождения памяти.

**4.5 Указатель this**

При вызове функции, принадлежащей классу, для обработки данных конкретного объекта этой функции автоматически передается указатель на тот объект, для которого функция вызвана. Этот указатель имеет фиксированное имя **this** и «тайно» определен в каждой функции класса следующим образом:

**имя\_класса \*const this = адрес\_объекта;**

Изменить указатель **this** нельзя, т.к. это константный указатель. Он является дополнительным (скрытым) параметром каждой нестатической компонентной функции. Объект, который адресуется указателем **this**, становится доступным внутри принадлежащей классу функции именно с помощью этого указателя.

При совпадении имени компонента класса с именем формального параметра принадлежащей классу функции, указатель **this** позволяет компилятору разобраться, к какому параметру относится сообщение (однако снять эту неоднозначность можно и другим способом – с помощью операции изменения видимости **::**).

Очень удобен указатель **this** при необходимости передать компонентной функции адрес определенного объекта класса, например, при организации списков, звеньями которого должны быть объекты класса (пример приведен в разделе 5).

**5 Примеры программ**

1. Класс «комплексное число». Компонентные данные: действительная часть, мнимая часть, статический компонент – счетчик количества объектов (чисел). Компонентные функции: получение данных класса, печать данных. Дружественная функция печати данных.

**#include <iostream>**

**using namespace std;**

**//описание класса**

**class complex**

**{**

**double real; //действительная часть**

**double imag; //мнимая часть**

**static int N; //статическое поле - счетчик числа объектов**

**public:**

**//описание конструктора класса**

**complex(double = 1.0, double = 0.0);**

**//описание компонентной функции печати**

**void display();**

**//описание дружественной функции печати**

**friend void print(complex);**

**//описание и определение функций доступа к компонентам**

**double getReal()**

**{**

**return real;**

**}**

**double getImag()**

**{**

**return imag;**

**}**

**static int getN()**

**{**

**return N;**

**}**

**};**

**//определение конструктора**

**complex::complex(double re, double im)**

**{**

**real = re;**

**imag = im;**

**N++;**

**}**

**//определение компонентной функции печати**

**void complex::display()**

**{**

**cout<<"real = "<<real;**

**cout<<", imag = "<<imag<<'\n';**

**}**

**//определение дружественной функции печати**

**void print(complex ob)**

**{**

**ob.display();**

**}**

**//инициализация статического компонента**

**int complex::N =0;**

**void main()**

**{**

**//создание объектов класса**

**complex x1,x2(3.0, 5.0);**

**//вызов компонентной функции печати**

**x1.display();**

**//вызов дружественной функции печати**

**print(x2);**

**//вызов компонетных функций доступа к полям класса**

**cout<<"real + imag = "<<x2.getReal()+x2.getImag()<<endl;**

**//вызов статической функции**

**cout<<"Defined "<<complex::getN()<<" numbers\n";**

**}**

Результат выполнения программы:

**real = 1, imag = 0**

**real = 3, imag = 5**

**real + imag = 8**

**Defined 2 numbers**

2. Создать класс, элементы которого формируют двусвязный список. Вывести содержимое списка на экран.

**#include <iostream>**

**using namespace std;**

**class member**

**{**

**static member \*last\_memb; //адрес последнего элемента**

**member \*prev; //на предыдущий элемент списка**

**member \*next; //на следующий элемент списка**

**char bukva; //значение элемента списка**

**public:**

**member(char); //конструктор**

**void add(); //добавление в конец списка**

**static void reprint(); //вывод списка на экран**

**};**

**//конструктор**

**member::member(char cc)**

**{**

**bukva=cc;**

**}**

**//добавление элемента в конец списка**

**void member::add()**

**{**

**if (last\_memb==NULL)**

**this->prev=NULL;**

**else**

**{**

**last\_memb->next=this;**

**this->prev=last\_memb;**

**}**

**last\_memb=this;**

**this->next=NULL;**

**}**

**//вывод списка на экран**

**void member::reprint()**

**{**

**member \*uk; //вспомогательный указатель**

**uk=last\_memb;**

**if (uk==NULL)**

**{**

**cout<<"\nThe list is empty!";**

**return;**

**}**

**else**

**cout<<"\nThe list:\n";**

**while (uk!=NULL) //печать списка в обратном порядке**

**{**

**cout<<uk->bukva<<'\t';**

**uk=uk->prev;**

**}**

**}**

**//инициализация статического указателя**

**member \*member::last\_memb=NULL;**

**void main()**

**{**

**member A('a'); //формирование объектов класса**

**member B('b');**

**member C('c');**

**member D('d');**

**member::reprint(); //вызов статической функции**

**A.add(); //включение объектов в двусвязный список**

**B.add();**

**C.add();**

**D.add();**

**member::reprint(); //печать списка в обратном порядке**

**}**

Результат выполнения программы:

**The list is empty!**

**The list:**

**d c b a**

**6 Контрольные вопросы**

1. Дайте определение класса в языке С++. Каков синтаксис его объявления?
2. Перечислите известные вам спецификаторы доступа к элементам класса. Для чего они используются?
3. Какой спецификатор доступа предполагается по умолчанию для классов? Сравните со структурами.
4. Как создать объект класса, массив объектов, указатель на объект?
5. Можно ли в объявлении класса описать только прототип функции, а ее реализацию вынести отдельно? Как это сделать?
6. Что такое дружественная функция? Для чего она может использоваться?
7. В каких случаях могут применяться статические данные класса? В чем их отличие от обычных данных?
8. Что такое конструктор класса? Обязательно ли он должен присутствовать?
9. Опишите известные вам способы описания конструкторов.
10. Дайте определение деструктора класса. Как его описать?

**7 Варианты заданий для самостоятельного решения**

1. Описать класс «точка на плоскости», предусмотрев в нем функции: «показать точку», «стереть», «переместить», «получить координаты точки». Построить отрезок между двумя точками, определенными как экземпляры класса «точка».

2. Определить класс «комплексное число», предусмотрев в нем функции получения мнимой части, получения действительной части, вывод всего комплексного числа в формате *ai+b*. Определить дружественную функцию вывода комплексного числа в экспоненциальной форме.

3. Определить класс **Date**, предусмотрев в нем компонентные данные: год, месяц, день; компонентные функции вывода даты на экран в формате «чч.мм.гг», получения каждого элемента данных по отдельности. Определить дружественную функцию, определяющую промежуток времени (в днях) между двумя событиями.

4. Составить двусвязный список на основе класса, объекты которого будут формировать этот список. В описание класса должны входить данные для хранения имени автора, названия книги, год издания, наименование издательства; функции для доступа к отдельным компонентам списка, вывод на экран элементов списка, добавление элемента в начало списка, удаление элемента списка. Использовать указатель **this**.

5. Определить класс **Time**, предусмотрев в нем компонентные данные: часы, минуты, секунды; компонентные функции вывода на экран времени в формате «чч-мм-сс», получения каждого элемента данных по отдельности. Определить дружественную функцию, прибавляющую к текущему времени одну секунду (учесть смену суток).

6. Определить класс «символ в заданной позиции экрана» (использовать модуль **conio.h**). Предусмотреть компонентные данные: координаты символа на экране, значение символа; компонентные функции: вывода символа на экран в заданной позиции, перемещения символа в новую позицию, получения текущих координат символа и самого символа. Создать дружественную функцию замены символа в текущей позиции.

7. Составить односвязный список на основе класса, объекты которого будут формировать этот список. В описание класса должны входить данные для хранения фамилии студента, номера группы, оценок по трем предметам; функции для доступа к отдельным компонентам списка, вывод на экран элементов списка, а также среднего балла студента, добавление элемента в конец списка, добавление элемента в начало списка, удаление элемента списка. Использовать указатель **this**.

8. Описать класс «окружность», содержащий данные о координатах центра окружности, ее радиусе и цвете. Предусмотреть компонентные функции получения каждого данного по отдельности, рисования, стирания, переноса окружности, изменения ее радиуса и цвета.

9. Определить два дружественных класса: «точка в трехмерном пространстве» и «отрезок в трехмерном пространстве». Компонентными данными второго класса являются координаты начала и конца отрезка – объекты класса точки, а также цвет объекта. Предусмотреть компонентые функции вычисления длины отрезка и его рисования. С помощью объектов описанных классов нарисовать на экране многоугольник.

10. Создать класс «символьная строка на экране» с компонентными данными: строка символов, длина строки, координаты строки. Строку создавать динамически. Предусмотреть компонентные функции: получения указателя на строку, ссылки на длину строки, позиции строки на экране, печати длины строки, печати самой строки в заданной позиции на экране, перемещения строки в другую позицию.

11. Описать класс окна (использовать модуль **conio.h**) с компонентными данными: координаты окна на экране, цвет фона, цвет символа, строка символов. Определить компонетные функции: получения каждого данного по отдельности, вывода окна на экран, изменения цветов фона и символа, изменения размера окна, вывода строки в окне.

12. Создать двусвязный список на основе класса, объекты которого будут формировать этот список. В описание класса должны входить данные для хранения фамилии, имени, года рождения и роста; функции для доступа к отдельным компонентам списка, вывод на экран элементов списка, добавление элемента в начало списка, удаление элемента списка. Использовать указатель **this**.

13. Описать класс «секундомер» с компонентными данными: исходное время, количество секунд для отсчета. Предусмотреть компонентные функции: получения и установки значений компонентных данных по отдельности, запуск секундомера, остановка, сброс секундомера.

14. Создать односвязный список на основе класса, объекты которого будут формировать этот список. В описание класса должны входить данные для хранения комплексных чисел; функции для доступа к отдельным компонентам списка, вывод на экран элементов списка в формате *ai+b*, добавление элемента в начало списка, удаление элемента списка. Использовать указатель **this**.

15. Определить класс «дробное число» с компонентными данными: целая часть, числитель, знаменатель (проверить, не равен ли знаменатель нулю). Предусмотреть компонентные функции: получения каждого элемента данных по отдельности, форматного вывода дроби на экран (например, «5 ½», если целая часть равна нулю, то не выводить ее), приведения дроби к несократимому виду. Описать дружественную функцию преобразования дроби в десятичный формат и ее вывода на экран.

16. Создать класс «многочлен» с компонентными данными: порядок многочлена, массив его коэффициентов. Определить компонентные функции вывода многочлена на экран в формате «*y=a0+a1x+a2x2+…anxn*», вычисления коэффициентов и вывода на экран его производной.

17. Описать класс «квадратное уравнение» с компонентными данными – его коэффициентами. Определить компонентные функции вывода уравнения на экран в формате «*ax2+bx+c=0*», нахождения и печати корней, вывода уравнения на экран в формате «*a(x-x1)(x-x2)=0*»(если имеются два корня) либо «*a(x-x1)2=0*», если один корень.

18. Создать класс «Форматированный вывод» с компонентными данными: элемент вывода (целое число, вещественное число или строка – предусмотреть различные виды конструкторов, если задано число, то преобразовать его в строку), ширина поля вывода, символ заполнения, выравнивание (по левому или по правому краю). Предусмотреть компонентные функции получения каждого элемента класса, изменения ширины поля вывода и типа выравнивания, вывода элемента на экран с учетом установленных параметров.

19. Определить класс «торговый терминал» с компонентными данными: оптовая цена товара, количество купленных единиц товара, торговая наценка, общая цена покупки, сумма, заплаченная покупателем, сумма сдачи. Описать компонентные функции:

* учет данных о покупке: вводится количество единиц товара и его оптовая цена, рассчитывается общая сумма покупки и осуществляется запрос на продолжение ввода;
* расчет с покупателем: вводится количество денег, уплаченных покупателем, далее либо выводится сообщение о том, что данной суммы недостаточно, либо рассчитывается сумма сдачи.

20. Описать класс «точка» со следующими компонентными данными: координаты точки, ее цвет, счетчик количества точек (статический компонент) – и функциями рисования точки на экране и определения количества точек. Создать в цикле несколько объектов класса (каждый раз запрашивать разрешение на продолжение ввода) и нарисовать их на экране. Каждую новую точку соединять с предыдущей отрезком. Последнюю точку соединить с первой. С помощью статического компонента определить, сколько углов имеет получившийся многоугольник.

21. Описать класс «треугольник» с компонентными данными: координаты вершин, длины сторон, цвет треугольника. В конструкторе осуществлять проверку, можно ли построить треугольник с заданными сторонами. Определить компонентные функции: получения каждого данного класса в отдельности, вычисления его площади и периметра, вывода его на экран. Создать дружественную функцию определения вида треугольника: равносторонний, равнобедренный, простой.

22. Создать класс «арифметическое выражение» с компонентными данными: 1-й аргумент, 2-й аргумент, знак арифметического выражения (перечислимого типа – ‘+’, ‘–‘, ’\*’, ‘/’). Определить компонентные функции: получение каждого элемента класса по отдельности, изменение значения операндов, вычисление значения выражения, вывод на экран в формате «**арг1 операция арг2 = результат**».

23. Описать класс «квадрат на плоскости» с компонентными данными: координаты центра квадрата, длина стороны, цвет. Предусмотреть компонентные функции: получение каждого элемента класса по отдельности, рисование квадрата, удаление, перемещение, изменение цвета.

24. Создать класс «студент» с компонентными данными: фамилия, курс, оценки по четырем предметам, базовая стипендия и коэффициент надбавки к стипендии (статические данные), общая сумма стипендии. Определить компонентные функции: вывод на экран данных о студенте, перевод студента на следующий курс, вычисление общей суммы стипендии исходя из его оценок (троечникам стипендия вообще не платится, хорошистам – базовая стипендия, отличникам – базовая стипендия, умноженная на коэффициент надбавки).

25. Описать класс «матрица чисел» с компонентными данными: две размерности, указатель на элементы матрицы. Предусмотреть компонентные функции: получение доступа к элементу, вывод матрицы на экран, изменение знака каждого элемента, умножение матрицы на число.

26. Создать класс «время» с компонентными данными: число, месяц, год. Определить компонентные функции: получить каждый элемент класса по отдельности, вывести значение времени на экран, добавить к данному времени *n* часов, добавить *n* минут, добавить *n* секунд. Описать дружественную функцию «переход на летнее время».

27. Создать класс «стек символов» с компонентными данными: максимальный размер стека и указатель на вершину стека (статические компоненты), область памяти (массив) для элементов стека. Предусмотреть компонентные функции: поместить символ в стек, извлечь символ из стека, проверить стек на пустоту, вывести содержимое стека на экран.

28. Создать класс «очередь целых чисел» с компонентными данными: максимальный размер очереди, указатели на начало и конец очереди (статические компоненты), область памяти (массив) для элементов очереди. Предусмотреть компонентные функции: поместить число в очередь, извлечь число из очереди, проверить очередь на пустоту, вывести содержимое очереди на экран.