**Практическое занятие №12**

**«Классы. Конструкторы и деструкторы, конструктор копии и статические элементы. Доступность компонентов класса: private, public. Дружественные функции (friend)»**

**Теория**

*Классы можно создавать как с помощью слова class, так и с помощью слова struct – разница будет только в спецификаторах доступа по умолчанию – об этом будет в следующем занятии о наследовании классов.*

Класс можно рассматривать как тип данных, но в отличие от обычного типа, определяющего набор значений, класс объединяет в себе данные и подпрограммы (функции) обработки этих данных. Класс можно рассматривать как дальнейшее развитие типа данных структура. В классе, как и в структуре, объединяются данные различных типов. Эти данные принято называть полями. Помимо этого, класс содержит функции, которые называют методами. Для объявления класса используется ключевое слово class, за которым следует имя типа (класса), а затем в фигурных скобках следуют объявления типа каждого поля и метода. Объявление завершается точкой с запятой.

class [имя типа (класса)]

{

[спецификатор доступа] тип1 поле1;

[спецификатор доступа] тип2 поле2;

…..

[спецификатор доступа] типN полеN;

[спецификатор доступа] заголовок метода1;

[спецификатор доступа] заголовок метода2;

……

[спецификатор доступа] заголовок методаM;

};

Например, объявление класса точка на плоскости с методами, позволяющими инициализировать поля, являющиеся координатами, и считывать значения этих полей, выглядит следующим образом:

class point

{

public:

float x,y;

void initx(float ix);

void inity(float iy);

float getx();

float gety();

};

Поля класса могут иметь любой тип, кроме типа этого же класса (могут быть указателями или ссылками на этот тип). Они могут объявляться с модификатором const, при этом они инициализируются только один раз и не могут изменяться. Инициализация полей при описании не разрешается.

Поля и методы класса объявляются с использованием спецификаторов доступа private, public, protected. Элементы, описанные после служебного слова private, доступны только внутри этого класса и недоступны вне этого класса. Этот вид доступа принят в классе по умолчанию. Элементы, описанные с использованием спецификатора public, доступны вовне этого класса (в области видимости объекта этого класса) и определяют интерфейс класса. Элементы, описанные в секции protected (защищенный), рассматриваются как открытые в производных классах (рассматриваются дальше в наследовании) и закрытые в остальной части программы.

Секции могут располагаться в произвольном порядке, причем каждая секция может создаваться любое количество раз. Действие очередного спецификатора распространяется до следующего спецификатора или до конца объявления класса. Рекомендуется использовать не более одного раздела каждого вида, группируя данные соответствующего раздела.

Конкретные переменные типа класс называют экземплярами класса, или объектами. Время жизни и видимость объектов зависят от вида и места их описания и подчиняются общим правилам языка.

Каждый объект получает уникальный набор полей, но общий для всех объектов класса набор методов. Поскольку данные (поля) и методы инкапсулированы в одном обекте, то все данные (поля) глобальны по отношению к любым его методам и могут использоваться с указанием простого имени. При этом формальные параметры методов совпадать с именами полей не должны.

Функции-члены класса имеют доступ ко всем членам класса, объявленным в классе. Функции-члены класса могут вызывать друг друга напрямую. Это становится возможным благодаря наличию специального параметра this, который хранит адрес области памяти переменной типа класс и неявно передается методам при вызове.

Методы класса могут размещаться внутри описания класса. Однако при создании сложных классов методы выносятся за пределы кдасса. При этом заголовок метода (прототип) указывается внутри класса. Для определения функции – члена класса вне класса используется оператор разрешения области видимости :: (два двоеточия). При этом перед кодом метода указывается имя класса, которому он принадлежат.

Программа 1. Программа, демонстрирующая работу с классом точка

class point

{

public:

float x,y;

void initx(float ix);

void inity(float iy);

float getx();

float gety();

};

int main()

{

point p1;

p1.initx(5.5);

p1.inity(-2.2);

printf("x=%5.1f y=%5.1f",p1.getx(),p1.gety());

\_getch();

return 0;

}

void point::initx(float ix)

{ x=ix; }

void point::inity(float iy)

{ y=iy; }

float point::getx()

{ return x; }

float point::gety()

{ return y; }

Классы могут быть глобальными (объявляются вне какого-либо блока) и локальными (объявляются внутри функции или другого класса).

В состав любого класса входят специальные методы – **конструктор и деструктор**, которые объявляются в классе или наследуются от базового класса. Конструктор используется для создания объекта (выделения памяти под него), деструктор предназначен для освобождения памяти. Конструктор вызывается каждый раз, когда создается объект: при объявлении экземпляра переменной автоматически вызывается конструктор для создания экземпляра. Если конструктор явно не определен, то вызывается конструктор по умолчанию без параметров. Конструктор определяет необходимый размер памяти под размещение объекта в динамической памяти, осуществляет выделение памяти и распределение объекта в памяти, помещает адрес этой памяти в экземпляр – переменную типа класса. Деструктор освобождает динамическую память и разрушает объект.

Класс может иметь несколько конструкторов для разной инициализации данных членов.

Существует ряд правил объявления конструктора и деструктоа. Имена конструктора и деструктора должны совпадать с именем класса, перед именем деструктора ставится знак тильда **~**. Конструктор и деструктор не имеют типа возвращаемого результата. Если не указывается ни одного конструктора, то он создается компилятором автоматически. Конструкторы используются в основном для инициализации полей.

Доступ к полям и методам объекта, объявленным со спецификатором public, извне класса выполняется с использованием операции прямого выбора (используется точка), например, p.x= 9.7; p.y=-3.4; p.getx(); Однако такой способ нежелателен. К закрытым полям доступ осуществляется с помощью вызова соответствующих методов.

**Основные принципы построения классов**

Основные принципы построения классов – инкапсуляция, наследование, полиморфизм.

Инкапсуляция означает объединение в виде одного целого (конструкции) данных и программ обработки этих данных. Этот принцип позволяет максимально изолировать объект класса от внешнего воздействия, что повышает надежность программ, использующих объекты. Создание библиотек классов значительно повышает производительность труда программистов по аналогии с использованием библиотек подпрограмм в прошлые времена.

Наследование означает возможность построения иерархии классов, т.е. порождать классы, являющиеся потомками родительских классов. При этом дочерний класс наследует поля и методы родительского класса, а также имеется возможность добавлять новые поля и методы и переопределять родительские поля и методы.

Принцип наследования позволяет реализовать последовательный процесс разработки классов. В результате использования принципа наследования получается часто ветвящееся дерево классов. Очередной потомок дополняется новыми полями и методами и передает их потомкам. В C++ реализована концепция множественного наследования – создание класса, имеющего несколько базовых классов.

При описании класса в его заголовке перечисляются классы, являющиеся для него базовыми. Возможность обращения к элементам этих классов регулируется с помощью ключей доступа private, protected, public:

Class имя\_класса :[private| protected | public] базовый класс

{тело класса};

По умолчанию для классов используется ключ доступа private.

Доступ к полям базового класса в зависимости от спецификатора доступа к базовому классу и спецификаторов полей базового класса определяется следующей таблицей.

Таблица 1 - Ключи доступа к полям базового класса при использовании слова **class**

|  |  |  |
| --- | --- | --- |
| Ключ доступа | Спецификатор в базовом классе | Доступ в производном классе |
| private | private  protected  public | нет  private  private |
| protected | private  protected  public | нет  protected  protected |
| public | private  protected  public | нет  protected  public |

Полиморфизм заключается в том, что методы классов на различных уровнях иерархии могут иметь одинаковые имена, но различное содержание. Одно и то же имя метода связывается с различными его реализациями на каждом уровне иерархии.Это действие используется всеми объектами иерархии, но в зависимости от типа объекта используется по-разному. Использование конкретного метода при обрашении к методу с заданным именем определяется типом объекта.

**Практика**

1. Создать класс комплексных чисел с функцией вывода **(lab12\_01.cpp).**
2. Создать класс книг с полями Автор, название книги, издательство, год издания **(lab12\_1.cpp)**
3. Создать класс книг с полями Автор, название книги, издательство, год издания с конструктором и деструктором **(lab12\_2.cpp)**
4. Классы могут содержать описание и других функций, которые становятся членами класса. Задание: изменить предыдущий пример таким образом, чтобы типизированные члены класса были скрыты, а для отображения класса существовала специальная функция **(lab12\_3.cpp)**
5. Идея объектного программирования состоит в том, что детальное программирование объектов осуществляется в одном месте, а их использование в другом. Задание: задать описание описание класса в заголовочном файле .h. Его детальную реализацию произвести в специальном модуле. Для этого использовать специальный элемент описания «::» - который говорит о том, что данный элемент относится к описанию класса. В вызывающей программе создать объекты указанного класса, использующие только описание класса **(lab12\_4.cpp)**
6. **Конструктор копии.** Рассмотрим уже знакомый нам пример, в котором создаются объекты книга. Определим ещё одну функцию, которая на вход будет принимать объект нашего типа. Произойдет ошибка, связанная с тем, что при вызове программы, наш объект должен скопироваться в стек. Если это не оговорено специально, то копирование производится байт за байтом. В том числе и адресные переменные будут скопированы вместе с адресом. Но при выходе из функции сработает деструктор и уничтожит содержимое адреса. Чтобы избежать возникновение этой ошибки, мы должны предпринять дополнительные действия **(lab12\_5.cpp)**
7. Копирование объекта в стек производится при помощи конструктора копии. Избежать указанную проблему можно, закрыв доступ к конструктору копии. В этом случае, мы не сможем передавать указанный объект в другую программу. В примере демонстрируется также, как можно обойти это ограничение, передав на вход адрес объекта **(lab12\_6.cpp)**
8. **Статические элементы.** Создать класс со статической переменной, которая подсчитывает общее количество создаваемых объектов **(lab12\_7.cpp)**
9. Статические функции также создаются один раз для всего класса. Они не связаны с конкретным объектом, но могут получать объекты во входном потоке **(lab12\_8.cpp)**
10. Создать класс point. В указанном классе, помимо координат включить int ID, которое должно содержать уникальный номер объекта. Этот номер должен присваиваться в любом конструкторе объекта. Для присвоения этого номера ввести статическую переменную, которая выдаёт этот номер, а затем увеличивается на 1.
11. Создать класс товаров с полями наименования товара, ценой, процентом наценки, функциями ввода и вывода. Представить поле процента наценки как статическое и для разных товаров присвоить разные проценты для розничных цен и вывести те, у которых процент равен 7, а также создать конструктор и деструктор для класса goods.
12. Создать класс point, содержащий две вещественные координаты точки (x, y), функции ввода и вывода координат объекта, а также функцию вычисления длины вектора (корень квадратный из квадрата координат). Создать массив объектов при помощи генератора случайных чисел и распечатать. Добавить конструктор со значениями по умолчанию и пустой деструктор.
13. **Функции friend.** Использование функции friend для вычисления суммы двух переменных файле **(lab12\_9.cpp)**
14. Использование friend для перегрузки левой операции (+). Важным способом использования описания **friend** является специальный способ перегрузки операторов. Обычный способ перегрузки предполагает, что перегружаемая функция относится к объекту, который находится слева от функции. Эта необходимость может возникнуть, если слева находится объект другого типа **(lab12\_10.cpp).**
15. Создать класс "Транспортное средство", в нем описать поля: наименование ТС, скорость ТС, пройденное расстояние, статические поля - максимальное расстояние MAX\_DISTANCE, максимальная скорость MAX\_SPEED, методы - получения информации о скорости, получения информации о текущем пробеге, получения информации о текущем времени эксплуатации, установки имени транспортного средства, конструктор с полями по умолчанию, деструктор. Описание класса производить в отдельном cpp файле с внешним определением функций класса.
16. Для класса complex из задания **(lab12\_01.cpp)** перегрузить (ввести дружественные) операции-функции сложения, вычитания, умножения, деления с некоторым введенным числом.