ЛАБОРАТОРНАЯ РАБОТА № 3
КЛАССЫ: ОСНОВНЫЕ ПОНЯТИЯ И ОПРЕДЕЛЕНИЯ

Цель работы – **изучить понятие класса, механизмы работы с классами, научиться обеспечивать вызов методов и обращение к полям классов, изменять видимость компонент в определении класса с использованием спецификаторов доступа, перегружать операции и использовать дружественные функции.**

Теоретические сведения

Данный раздел содержит краткий обзор сведений, необходимых для начала работы в стиле объектно-ориентированного программирования. Поэтому после ознакомления с ним рекомендуется обратиться к учебникам [3] или [4].

Классы и объекты

Класс – это производный структурированный тип, введенный программистом на основе существующих типов. Механизм классов позволяет создавать типы в соответствии с принципами абстракции данных, т.е. класс задает некоторую структурированную совокупность типизированных данных и позволяет определить набор операций над этими данными.

Простейшим образом класс можно определить с помощью конструкции:

```
class имя_класса
{
```

```
    список_компонентов;
};
```

где *class* – служебное (ключевое) слово, *имя_класса* – произвольно выбираемый идентификатор, *список_компонентов* – определения и описания принадлежащих классу данных и функций.

Компонентами класса могут быть данные, функции, перечисления, битовые поля, дружественные функции, дружественные классы и имена типов.

Данные, содержащиеся внутри класса, называются **полями класса**, ***данными-членами*** или ***компонентными данными***. Принадлежащие классу функции называются **методами класса**, ***функциями-членами*** или ***компонентными функциями***.

Переменная пользовательского типа, описанного классом, называется **объектом** или **экземпляром класса**. Для объявления объекта класса используется конструкция:

```
имя_класса имя_объекта;
```

Для доступа к компонентам конкретного объекта заданного класса используется уточненное имя:

```
имя_объекта.имя_поля /* для поля данных */
имя_объекта.имя_метода() /*для обращения к функции-члену класса */
```

Другой способ доступа к элементам объекта некоторого класса предусматривает явное использование указателя на объект класса и операции косвенного выбора компонента

```
указатель_на_объект_класса -> имя_элемента

указатель_на_объект_класса -> обращение_к_функции().
```

Спецификаторы доступа

Для управления видимостью компонент в определении класса можно использовать спецификаторы доступа. Определены следующие спецификаторы доступа:

1) ***private*** – собственный, частный, недоступны для внешних обращений;

1) ***protected*** – защищенный, используется при построении иерархии классов;

2) ***public*** – общедоступный.

Для сокрытия данных внутри объектов класса достаточно перед их описанием в определении типа поместить требуемый спецификатор. Как правило, поля данных помещают в скрытую часть класса, а доступ к ним осуществляют только через методы. По умолчанию все компоненты класса являются скрытыми (private).

Компонентные данные

Определение полей класса внешне аналогично обычному описанию переменных, однако их нельзя инициализировать. Для инициализации данных используется специальный метод – конструктор.

Существуют различия между обращениями к данным класса из функций-членов данного класса и из других частей программы. Принадлежащие классу функции имеют полный доступ ко всем его

полям, то есть для обращения к данным достаточно указать только имя поля.

Для доступа к данным класса из операторов, выполняемых вне определения класса, непосредственное использование имен элементов не допустимо. Если объявление поля находится после ключевого слова private или protected, то обратиться к нему «извне» невозможно. Для обращения к открытым (public) полям используется уточненное имя: *имя_объекта.имя_поля* или *указатель_на_объект->имя_поля*.

Компонентные функции

Компонентная функция может быть определена как в теле класса, так и вне его. В последнем случае в описание класса необходимо поместить ее прототип.

При определении компонентной функции вне класса программист должен сообщить компилятору, к какому именно классу она относится. Для этого используется бинарная операция «***::***», называемая *операцией глобального разрешения* или *операцией расширения области видимости*. Формат ее использования:

```
    тип            имя_класса            ::            имя_функции
( список_формальных_параметров )
    {
        тело_функции;
    }
```

Приведенная конструкция называется квалифицированным именем компонентной функции и означает, что функция

принадлежит данному классу и лежит в его области видимости. Именно такое описание привязывает функцию к классу и позволяет в ее теле непосредственно использовать любые данные и функции класса.

Пример класса:

```
class point
{
      int x, y; /* поля данных; по умолчанию private */
   public:
      void set_x (int a) { x=a; } /* установка значения поля x */
      void set_y (int a) { y=a; } /* установка значения поля y */
      int get_x(); /* прототип функции получения значения поля x */
      int get_y(); /* прототип функции получения значения поля y */
};
int point::get_x() /*внешнее определение функций */
{ return x; }
int point::get_y()
{ return y; }
```

Конструкторы и деструктор

В классе всегда явно или неявно присутствуют специальные методы, которые называются **конструктором** и **деструктором**. Конструктор выполняется автоматически в момент создания объекта (при определении переменной-объекта или выделении памяти под объект с помощью оператора new), деструктор – при

его уничтожении (завершении времени жизни переменной или освобождении памяти с помощью оператора delete).

По умолчанию в классе всегда автоматически формируется конструктор без параметров, который только создает объект. Помимо создания объекта конструктор может выделять память под данные, хранимые в объекте, и инициализировать поля объекта. Различают три типа конструкторов: *конструктор без параметров*, *конструктор с параметрами* и *конструктор копирования*. У конструкторов не бывает возвращаемого значения, а имя всегда совпадает с именем класса. Формат определения конструктора в теле класса таков:

```
имя_класса (список_формальных_параметров)
{
    операторы тела конструктора;
}
```

При отсутствии параметров скобки остаются пустыми.

Параметрам конструктора можно присвоить значения, которые будут использоваться, если пользователь не укажет их при объявлении объекта. Эти значения называются значениями по умолчанию. В классе может быть несколько конструкторов, но только один с умалчиваемыми значениями параметров.

Для вышеобъявленного класса point конструктор с параметрами можно объявить так:

```
point (int a=0, int b=0) /* по умолчанию значения параметров равны 0 */
{
      x = a;
      y = b;
}
```

При объявлении объектов данного класса:

```
point A, B(7,6);
```

поля объекта A получат значения по умолчанию (нули), а поля объекта B станут равными значениям параметров (x=7, y=6).

Конструктор копирования – это специальный вид конструктора, получающий в качестве единственного параметра ссылку на объект этого же класса. Общий вид: T :: T(T&) {...}, где T – имя класса.

Этот конструктор вызывается в тех случаях, когда новый объект создается путем копирования существующего:

- при описании нового объекта с инициализацией другим объектом;
- при передаче объекта в функцию по значению;
- при возврате объекта из функции.

Конструктор копирования, создаваемый компилятором автоматически, выполняет побайтное копирование объекта. Если при создании объекта не должна динамически выделяться память, переопределять его не требуется.

Динамическое выделение памяти для объектов какого-либо класса создает необходимость не только в переопределении конструктора копирования, но и в освобождении этой памяти при уничтожении объекта. Такую возможность обеспечивает специальный компонент класса – деструктор (разрушитель объектов) класса. Для него предусматривается стандартный формат определения:

```
~имя_класса ()
{
    операторы_тела_деструктора;
}
```

Название деструктора всегда начинается со специального символа «тильда», за которым без пробелов или других разделительных знаков помещается имя класса. У деструктора не может быть параметров (даже типа void), и он не имеет возвращаемого значения. Вызов деструктора выполняется неявно, автоматически, как только объект класса уничтожается.

Перегрузка операций

Перегрузка операций дает возможность использовать стандартные знаки операций для выполнения действий над объектами создаваемого класса, например, описывая класс «Длинное целое» логично знаком «+» обозначить операцию сложения. Приоритет операций при перегрузке не изменяется.

Имя метода, переопределяющего операцию, складывается из ключевого слова operator и знака операции после него. Если перегружаемая операция является унарной, то параметров у этого метода не будет, так как действия будут производиться над текущим объектом. Если перегружаемая операция бинарная, то параметром будет являться второй операнд. Тип результата, возвращаемого функцией, зависит от характера операции. Например, операция сравнения на равенство двух объектов класса point будет определяться так:

```
int operator== (point &p)
{
   return x==p.x && y==p.y;
}
```

Компилятором автоматически перегружается операция присваивания, которая выполняет побайтное копирование объекта. Если объект не использует динамически выделенную память, переопределять операцию присваивания не требуется.

Указатель this

Когда функция, принадлежащая классу, вызывается для обработки данных конкретного объекта, этой функции автоматически и неявно передается указатель на тот объект, для которого функция вызвана. Этот указатель имеет фиксированное имя ***this*** и неявно определен в каждой функции класса следующим образом:

```
имя_класса * const this = адрес_обрабатываемого_объекта.
```

Имя this является служебным словом, явно описать или определить его нельзя. Так как этот указатель является константным, изменять его нельзя, однако в каждой принадлежащей классу функции он указывает именно на тот объект, для которого функция вызывается.

Указатель this удобно использовать в тех случаях, когда функция должна вернуть значение объекта, для которого она вызвана. Например, операция увеличения полей объекта класса point в m раз:

```
point operator*= (unsigned m)
{
    x *= m;
    y *= m;
    return *this;
}
```

Дружественные функции

Дружественной функцией класса называется функция, которая, не являясь его компонентом, имеет доступ к его защищенным и собственным компонентам. Такие функции используются, когда необходимо одновременно обращаться к скрытым компонентам объектов разных классов. Для получения прав друга функция должна быть описана в теле класса со спецификатором ***friend***.

Например, для вывода значений полей объекта рассматриваемого класса point можно перегрузить оператор сдвига в потоке вывода *ostream*:

```
/* объявление функции в теле класса point*/
friend ostream &operator << (ostream &stream, point &p);
/* определение функции вне всех классов */
ostream &operator << (ostream &stream, point &p)
{
    stream << "(" << p.x << ", " << p.y << ")";
    return stream;
}
```

Постановка задачи

Описать класс в соответствии с индивидуальным вариантом задания и реализовать все его методы. Каждый класс помимо указанных в варианте методов должен содержать конструктор с параметрами, конструктор копирования, деструктор, методы ввода с клавиатуры, установки и получения значений полей, вывода этих значений на экран. В каждом методе класса, включая конструкторы и деструктор, предусмотреть отладочную печать сообщения, содержащего имя метода. Написать программу для тестирования всех методов класса, выбор метода должен осуществляться с помощью меню.

Варианты заданий

Вариант 17.

Класс «Связанный вектор». Поля: координаты начала и конца вектора. Методы: вычисление длины вектора, угла между двумя векторами, перегрузка операции «||» как определение коллинеарности двух векторов,

операции сложения «+» двух векторов и расширенной операции присваивания «+=».