1. 函数重载

如果同一作用域内的几个函数名字相同但形参列表不同，那么我们称之为重载函数。

如：

void print(int i)

{

cout << "整数为: " << i << endl;

}

void print(double f)

{

cout << "浮点数为: " << f << endl;

}

void print(string c)

{

cout << "字符串为: " << c << endl;

}

这些函数接受的形参类型不一样，但执行的操作非常类似，当调用这些函数时，编译器会根据传递的实参类型推断想要的是那个函数。

虽然我们定义的这两个函数不相同，但是他们有共同的名字，编辑器会根据实参类型选择不同的函数。

对于重载函数来说，他们应该在形参数量或形参类型上有所不同，因为编译器以此来对函数进行选择。不允许两个函数除了返回值类型外其它所有的要素都相同。

1. 类的定义及实例化访问

定义一个类，本质上是定义一个数据类型的蓝图。这实际上并没有定义任何数据，但它定义了类的对象包括了什么，以及可以在这个对象上执行哪些操作。

类的定义

类定义是以关键字 class 开头，后跟类的名称。类的主体是包含在一对花括号中。类定义后必须跟着一个分号或一个声明列表。例如，我们使用关键字 class 定义 Box 数据类型，如下所示：

class Box

{

public:

double length; // 盒子的长度

double breadth; // 盒子的宽度

double height; // 盒子的高度

};

类的实例化对象

类提供了对象的蓝图，所以对象是根据类来创建的。声明类的对象，就像声明基本类型的变量一样。下面的语句声明了类 Box 的两个对象：

Box Box1; // 声明 Box1，类型为 Box

Box Box2; // 声明 Box2，类型为 Box

类的数据成员访问

在类中定义了若干个公有变量，要想对公有变量进行访问，必须先实例化然后使用直接成员访问运算符 (.) 来访问。私有的成员和受保护的成员不能使用直接成员访问运算符 (.) 来直接访问。

例：

Box1.length=10；

1. 类的成员函数

类的成员函数是指那些把声明写在类内部的函数，就像类定义中的其他变量一样。类成员函数是类的一个成员，它可以操作类的任意对象，可以访问对象中的所有成员。成员函数常常用来访问一些受保护的数据。

成员函数可以定义在类定义内部，或者单独使用范围解析运算符 :: 来定义。

在面向对象的程序设计中，我们常常单独使用范围解析运算符 :: 来定义成员函数。

举例：

直接在类的内部声明定义：

class Box

{

public:

double length; // 长度

double breadth; // 宽度

double height; // 高度

double getVolume(void)

{

return length \* breadth \* height;

}

};

单独使用范围解析运算符 :: 定义

class Box

{

public:

double length; // 长度

double breadth; // 宽度

double height; // 高度

double getVolume(void);// 返回体积

};

double Box::getVolume(void)

{

return length \* breadth \* height;

}

访问类成员函数的方式是用成员访问运算符 (.)

如：

Box Box1； //创建对象，将类实例化

Box. getVolume(); //调用对象的公有成员函数

1. 类访问修饰符

数据封装是面向对象编程的一个重要特点，它防止函数直接访问类类型的内部成员。类成员的访问限制是通过在类主体内部对各个区域标记 public、private、protected 来指定的。关键字 public、private、protected 称为访问修饰符。

一个类可以有多个 public、protected 或 private 标记区域。每个标记区域在下一个标记区域开始之前或者在遇到类主体结束右括号之前都是有效的。成员和类的默认访问修饰符是 private。

举例：

class Base

{

public:

// 公有成员

protected:

// 受保护成员

private:

// 私有成员

};

1. 公有成员

公有成员在类的外部是可访问的。

class Box

{

public:

double length; // 长度

double breadth; // 宽度

double height; // 高度

double getVolume(void);// 返回体积

};

double Box::getVolume(void)

{

return length \* breadth \* height;

}

Box Box1； //创建对象，将类实例化

Box. getVolume(); //在外部直接调用对象的公有成员函数

1. 私有成员 private

私有成员变量或函数在类的外部是不可访问的，甚至是不可查看的，无法像公有成员一样在外部直接访问，只有类和友元函数可以访问私有成员，这么做更多是出于安全方面的考虑。但有时我们也需要获取私有成员的数据，为了实现这一功能一般会在私有区域定义数据，在公有区域定义相关的函数来访问私有数据，以便在类的外部也可以调用这些函数。默认情况下，类的所有成员都是私有的。

举例：

class Box

{

double width; //未指定访问修饰符，默认为私有成员，无法被外部直接访问。

public:

double length;

void setWidth(double wid);

double getWidth(void);

};

double Box:: getWidth (void)

{

return width;

}

Box Box1； //创建对象，将类实例化

Box1. width; //报错，在类的外部无法直接访问私有成员

Box. getWidth (); //在外部直接调用对象的公有成员函数

//以此间接访问私有成员with的值

1. 保护成员 protected

保护成员变量或函数与私有成员十分相似，唯一不同是，保护成员在派生类（即子类）中是可访问的，在后续讲解类的继承时会做详细介绍。

1. 类的构造函数

类的构造函数是类的一种特殊的成员函数，它会在每次创建类的新对象时执行。如果没有显式声明，它也会隐式声明，实例化对象时也会调用，只不过不会进行其他人为的操作。

构造函数的名称与类的名称是完全相同的，并且不会返回任何类型，也不会返回 void。

构造函数常常被用来初始化，不仅可以对变量赋值，还可以调用函数。

利用构造函数进行初始化时，其他类成员声明的先后顺序不会影响到构造函数。

举例：

class Line

{

public:

void setLength(double len);

double getLength(void);

Line(); // 这是构造函数

private:

double length;

};

// 成员函数定义，包括构造函数

Line::Line(void)

{

cout << "Object is being created" << endl;

}

void Line::setLength(double len)

{

length = len;

}

double Line::getLength(void)

{

return length;

}

int main()

{

Line line; // 实例化对象

line.setLength(6.0); // 设置长度

cout << "Length of line : " << line.getLength() << endl;

return 0;

}

1. 可以对构造函数传递参数，用于为某些成员变量设置初始值

举例：

class Line

{

public:

void setLength(double len);

double getLength(void);

Line(double len); // 带有参数的构造函数

private:

double length;

};

// 成员函数定义，包括构造函数

Line::Line(double len) // 带有参数的构造函数

{

cout << "Object is being created" << endl;

length = len;

}

void Line::setLength(double len)

{

length = len;

}

double Line::getLength(void)

{

return length;

}

int main()

{

Line line(10); // 实例化对象，并传递参数

cout << "Length of line : " << line.getLength() << endl;

system("PAUSE");

return 0;

}

1. 使用初始化列表来初始化字段

在上述例子中我们使用构造函数来初始化变量

Line::Line(double len) // 带有参数的构造函数

{

cout << "Object is being created" << endl;

length = len;

}

我们还可以使用初始化字段的方式来初始化变量，两种方法是等价的

Line::Line(double len) : length(len)

{

cout << "Object is being created, length = " << len << endl;

}

如果有多个变量需要初始化需要用逗号将它们隔开

Line::Line(double len) : length(len)，width(wid)

{

cout << "Object is being created, length = " << len << endl;

}

1. 类的析构函数

类的析构函数是类的一种特殊的成员函数，它会在销毁对象时执行。如果没有显式声明，它也会隐式声明，在销毁对象时也会调用，只是不会进行其他人为操作。

析构函数的名称与类的名称是完全相同的，只是在前面加了个波浪号（~）作为前缀。它不会返回任何值，也不能带有任何参数，析构函数有助于在跳出程序（比如关闭文件、释放内存等）前释放资源。

举例：

class Line

{

public:

void setLength(double len);

double getLength(void);

Line(); // 构造函数声明

~Line(); // 析构函数声明

private:

double length;

};

// 成员函数定义，包括构造函数

Line::Line(void)

{

cout << "Object is being created" << endl;

}

Line::~Line(void)

{

cout << "Object is being deleted" << endl;

}

void Line::setLength(double len)

{

length = len;

}

double Line::getLength(void)

{

return length;

}

// 程序的主函数

int main()

{

Line line;

// 设置长度

line.setLength(6.0);

cout << "Length of line : " << line.getLength() << endl;

system("PAUSE");

return 0;

}

此程序的运行结果为：

Object is being created

Length of line : 6

但是，细心的你会发现在关闭程序的瞬间会出现这么一行字：

Object is being deleted

这是因为在程序结束时需要销毁对象，调用了对象的析构函数。

作业：

1. 创建桌子类，它包含桌子的材料、腿的数量、长、宽、高、颜色，对这些数据进行合理的初始化和存储，并给提供给外界合理的访问方式。
2. 现有游戏账户管理系统，它包含的信息有游戏ID、游戏等级、身份证号、游戏币数量，请定义一个类，对这些数据进行合理的初始化和存储，并给提供给外界合理的访问方式。