用类去定义对象时，系统会为每一个对象分配存储空间。如果一个类包括了数据和函数，要分别为数据和函数的代码分配存储空间。

按理说，如果用同一个类定义了10个对象，那么就需要分别为10个对象的数据和函数代码分配存储单元，如图8.4所示。

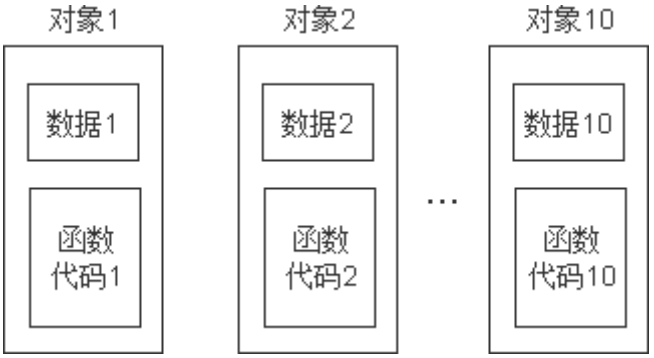

图8.4

能否只用一段空间来存放这个共同的函数代码段，在调用各对象的函数时，都去调用这个公用的函数代码。如图8.5所示。

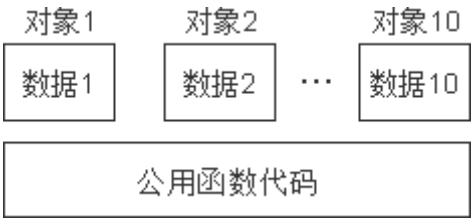

图8.5

显然，这样做会大大节约存储空间。C++编译系统正是这样做的，因此每个对象所占用的存储空间只是该对象的数据部分所占用的存储空间，而不包括函数代码所占用的存储空间。如果声明了一个类：

```
1. class Time
2. {
3.    public:
4.    int hour;
5.    int minute;
6.    int sec;
```

```
7.   void set( )
8.   {
9.     cin>>a>>b>>c;
10.   }
11. };
```

可以用下面的语句来输出该类对象所占用的字节数：

```
cout<<sizeof(Time)<<endl;
```

输出的值是12。

这就证明了一个对象所占的空间大小只取决于该对象中数据成员所占的空间，而与成员函数无关。

函数代码是存储在对象空间之外的。如果对同一个类定义了10个对象，这些对象的成员函数对应的是同一个函数代码段，而不是10个不同的函数代码段。需要注意的是，虽然调用不同对象的成员函数时都是执行同一段函数代码，但是执行结果一般是不相同的。

不同的对象使用的是同一个函数代码段，它怎么能够分别对不同对象中的数据进行操作呢？

原来C++为此专门设立了一个名为this的指针，用来指向不同的对象。需要说明：

1. 不论成员函数在类内定义还是在类外定义，成员函数的代码段都用同一种方式存储。
2. 不要将成员函数的这种存储方式和inline(内置)函数的概念混淆。不要误以为用inline声明(或默认为inline)的成员函数，其代码段占用对象的存储空间，而不用inline声明的成员函数,其代码段不占用对象的存储空间。不论是否用inline声明，成员函数的代码段都不占用对象的存储空间。用inline声明的作用是在调用该函数时，将函数的代码段复制插人到函数调用点，而若不用inline声明，在调用该函数时，流程转去函数代码段的人口地址，在执行完该函数代码段后，流程返回函数调用点。inline与成员函数是否占用对象的存储空间无关，它们不属同一个问题，不应搞混。
3. 应当说明，常说的“某某对象的成员函数”，是从逻辑的角度而言的，而成员函数的存储方式，是从物理的角度而言的，二者是不矛盾的。