面向对象程序设计(Object-Oriented Programming，简称 OOP)，主要是将“数据”和“程序”整合起来，使得软件的组件(Component 或 Module)更完整、独立，软件就更有弹性、更易于维护。这数据和程序的整合体，就称为“对象”(object)。对象是完美的组件，对象之间具有高度的独立性，所以维护软件时，极易调换对象。

OOPC(Object-Oriented Programming in C)是指 OOP 与 C 语言的结合，借助 C 语言的宏指令定义出 OOP 概念的关键字(Key Word)，C 程序员就能运用这些关键字来表达 OOP 概念，如类、对象、消息、继承和接口等。以 C 编写出来的面向对象程序统称为 OOPC 程序。

虽然 OOPC 程序的语法不像 C++那么简洁，但是 OOPC 也有其亮丽的特色，就是编译后的程序所占内存空间(Size)比 C++程序来得小，较能满足像嵌入式系统等的内存限制，程序员也较能有效调整程序的瓶颈而提升其执行速度。本节从最基础的对象传递消息(massage-passing)开始，说明 C 语言与 OOP 的特色及妙用，让您不必使用 C++也能享受 OOP 的特色。首先从对象(Object)的行为谈起。

1.13.1 对象的行为(Behavior)

树林中的树会长高或变矮(遇台风等)，电脑中储存这些数据的对象必须随之改变，即对象内的数据会改变。果树的果实售价改变了，Fruit_tree 类的对象也须改变；这种对象内数据的变化是对象的“行为”(Behavior)。对象的基本行为包括：

(1) 把数据送入对象并储存起来。

(2) 改变对象内的数据。

(3) 拿对象数据来做运算。

(4) 从对象中送出数据。

例如，打 8 折洗衣机的售价金额为多少？此时，可拿对象 a 内的价格数据来做运算，如图 1-35 所示。

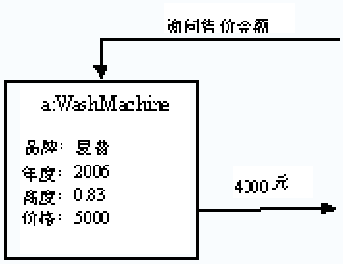


图 1-35　对象 a 的行为 1

我们的目的是要拿对象 **a** 中的价格数据**(5000** 元**)**和折扣**(0.8)**相乘，进而算出其售价金额**(4000** 元**)**。然而在 **OOP** 的观念中，则必须将其解释为“把‘询问售价金额？’送进对象 **a**，在对象内部做相乘运算，然后把售价 **4000** 送出来。”

其中，**4000** 是在对象 **a** 接受到外界的“消息”**(Message)**之后，才做出的反应，至于它的反应过程**(**乘法运算**)**则是在对象 **a** 内部完成的。就如同一个电灯泡，当电流进入灯泡中，此灯泡会发出亮光，这是人们日常唾手可得的经验。当你到自动售票机上买票时，只需把钱投入售票机中，经过一些处理之后，就有反应：把票吐出来，如图 **1-36** 所示。

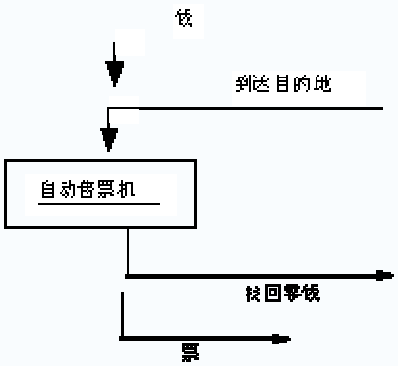


图 1-36　自动售票机的行为

上述实物对象**(**如自动售票机**)**接到消息**(Message)**时，其内部产生运作，并输出运作的结果。同样地，程序内的对象接到消息时，其内部也对数据做运算，并输出运算结果。例如，想知道洗衣机的高度为多少？可将此消息送进对象 **a** 中，它会输出此机器的高度，如图 **1-37** 所示。

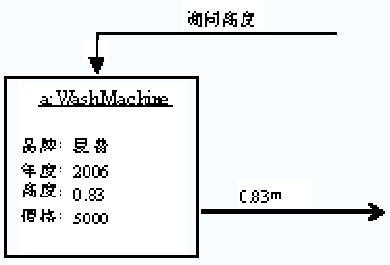


图 1-37　对象 a 的行为 2

对象对消息会产生反应，但并非对任何消息都有反应。例如，灯泡只会对电流有所反应——发光。自动售票机必须投入钱才会有反应——输出票来。如同一只猫，它对身边的一个馒头可能不会有反应，但对身旁的老鼠将会出现强烈的反应。

当我们使用灯泡或自动售票机时，能轻易学会其使用方法——知道输入什么消息，也很清楚它们的反应。类似地，在 OOP 程序中，您也能轻易学会对象的使用方法——知道输入什么消息，及了解其反应。所以，使用程序内的对象，就像使用灯泡一样简单方便。

换个角度来说，如果您是自动售票机的设计人，就须负责设计机内的处理过程(反应过程)，并使输入消息更简单，而反应更清楚。同理，如果您是对象的设计师，就得负责设计对象内的运算过程(对消息的反应过程)，并使输入消息更简单，而反应更清楚。

1.13.2 消息与运算

如果您是手表的设计人，则需把电池放入手表中以提供电力。此时，您既使用现成的对象电池，也创造新对象手表。同样情况，写电脑程序时，您经常是对象的使用人，也为对象的设计人。因此，利用既有而完美的对象去创造更有用的对象，这是 OOP 程序员的天职。设计手表时，您心里必定很清楚这新手表会有何功能，譬如：如何设定时间、表示时间(指针或数字)等。也就是说，您一定对这手表将呈现的“行为”(Behavior)有很清晰的构想。其行为包括：

(1) 它接受何种“消息”(Message)？例如按键输入时间。

(2) 对消息将会有何反应？例如显示时间或日期等。

在我们周围的实物世界中，各物体都有其固定的行为，所以我们能轻易掌握它，且觉得可爱。例如抽烟者常携带的打火机，只有用力按它时才有火苗出现；一颗手榴弹也在某种动作下，才会引爆。否则您一定不敢用打火机，面对敌人时也不敢用手榴弹。

设计程序中的对象时，也得设计它的“行为”，决定它将接受何种消息，并且对消息会产生何种反应。因为您是设计人，所以有项必须担任的工作——设计对象内部的运作，使它对消息产生正确的反应。

就像您组织手表内部的部件、打火机内部的结构以及手榴弹内部的引爆过程等。这就是对象设计者的主要工作，其目的是让别人有个好用而易于掌握的对象。

当您对所设计的对象的行为有清晰的构想之后，就自然知道应将哪些函数置于对象中。就如同设计手表者根据手表的功能决定应有哪些部件一般。您去购买现成的部件，创造部件，选择适当的电池，接下来就将这些部件组织在手表内。

编写程序时，您使用既有的函数(例如 C 已提供的函数库)，创造的新函数，也利用既有的对象，接下来就将这些函数和对象组织在新对象中。至于应如何组织这些函数？如何运用现成的对象？即为本小节的主题了。现在请回到 WashMachine 类的例子，上节里已经定义对象 a，如图 1-38 所示。

图 1-38　定义对象 a

接下来，可设计一个函数 computeSalePrice()，它能计算售价金额。如果把这函数加入对象 a 中，那么对象 a 就有反应，如图 1-39 所示。

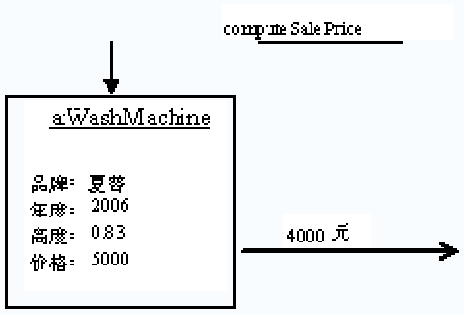


图 1-39　使用函数 computeSalePrice 来计算

如果另外设计一个函数 inquireHeight()，并将其加入对象 a 中，则对象 a 就能输出洗衣机的高度，如图 1-40 所示。

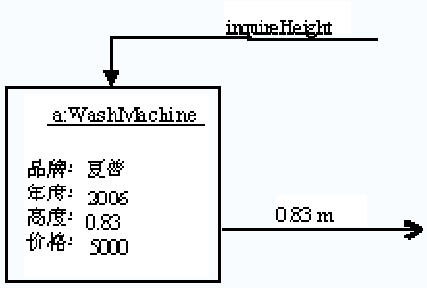


图 1-40　使用 inquireHeight 函数来计算

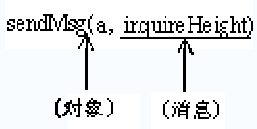
如同设计灯泡者将钨丝及稀有气体放入其中，灯泡才会发光。至于灯泡的使用人就不必为灯泡内部的东西及运作过程而伤脑筋了，因此，人们会喜欢使用灯泡。灯泡把材料和反应的过程“封装”**(Encapsulate)**在灯泡内，您只需把电流**(**消息**)**传入灯泡**(**对象**)**中，它就发亮**(**反应**)**。

编写程序时，您把函数摆入对象中，对对象内的数据做运算，并且输出结果，此时函数内的指令担任这项运算工作。由于函数存在对象内，而数据的运算在函数内进行，所以，运算的过程**(**即反应的过程**)**就被“封装”在对象之内。对象的使用人不必为这些函数的运算而伤脑筋，也就觉得对象是简单好用。如果电脑软件是由对象组织而成，人们就会觉得软件是既简单又好用。就像灯泡可装在车子上，也可装在房子内，并构成更大的对象**(**车子、房子都是对象**)**。此程序设计观念，就称为“面向对象程序设计”。

假设您已把 **computeAmount()**和 **inquireHeight()**两个函数加入对象 **a** 中，就可把消息送给对象 **a**，指令可写为：

```
sendMsg(a, computeSalePrice )
```

意思是把消息 **computeSalePrice** 送给对象 **a**。此时对象 **a** 内部的 **computeSalePrice()**函数就会进行运算**(**把 **5000** 和 **0.8** 乘起来**)**，并且送出售价金额 **4000** 元。您可输入另一消息，指令为：



此时，对象 a 内的 inquireHeight()函数就会进行运算(读取高度 0.83)并且发送该机器的高度，即 0.83m。

1.13.3 着手编写 OOPC 程序

1. 借助于 Laurent Deniau 定义的 C macros

在 2001 年，Laurent Deniau 以 C 的宏指令(Macro)定义出 OOPC 的机制，并在其网页上公开与大家分享。请参考其网页：

http://ldeniau.home.cern.ch/ldeniau/html/oopc/oopc.html

在本章里将以例子逐步说明如何使用 Laurent Deniau 的 OOPC 宏指令，并借之而编写出面向对象的 C 程序。像刚才所用到的 sendMsg 函数名称就是 Laurent Deniau 借助 C 宏指令而定义出来的。而您就能用它来表达面向对象的“消息传递”(Message-Passing)概念了。

2. 一个简单的 OOPC 程序

下述的 machine.h 文件定义 machine 类，machine.c 则是 machine 类的实现内容。

----范例 ex1-01(一)----

```
/*   machine.h   */
#ifndef  MACHINE_H
#define  MACHINE_H
#include  <ooc.h>
#undef  OBJECT
#define  OBJECT  machine
/* Object  interface  */
BASEOBJECT_INTERFACE
  char  const*  private(name);
  char  const*  private(year);
  float  private(height);
  float  private(price);
```

```
BASEOBJECT_METHODS
  float constMethod(computeSalePrice);
  float constMethod(inquireHeight);
  void constMethod(printName);
ENDOF_INTERFACE
/* Class interface */
CLASS_INTERFACE
  t_machine*const classMethod_(new)
        char const name[], char const year[], float height, float price __;
  void method_(init)char const name[], char const year[],
float height, float price __;
  void method_(copy)t_machine const*const pobj __;
ENDOF_INTERFACE
#endif
```

Laurent Deniau 借助 **C** 宏指令定义了 **OOPC** 的机制，并存在 **ooc.h** 头文件中。例如上述**#include <ooc.h>** 就包含了 **Laurent Deniau** 所定义的 **OOPC** 宏了。在类定义中使用**#define OBJECT machine** 宏来说明所定义的类名称 **machine**。接着使用指令：

```
BASEOBJECT_INTERFACE
  char const* private(name);
  char const* private(year);
  float private(height);
  float private(price);
```

说明此 **machine** 类有 **4** 个属性：**name**、**year**、**height** 和 **price**。

还有指令：

```
BASEOBJECT_METHODS
```

```
float constMethod(computeSalePrice);
float constMethod(inquireHeight);
void constMethod(printName);
```

说明此 **machine** 类所含的对象函数有 **computeSalePrice()**、**inquireHeight()**和 **printName()**。

指令如下：

```
CLASS_INTERFACE
t_machine*const classMethod_(new)
    char const name[], char const year[],
    float height, float price __;
void method_(init)
    char const name[], char const year[],
    float height, float price __;
void method_(copy)
    t_machine const*const pobj __;
```

则为 **machine** 类定义了 **3** 个类函数 **new()**、**init()**和 **copy()**。对象函数与类函数的区别在于：类函数属于类的，不属于任何对象，也就是不用来存取某对象内的数据值，通常只用来诞生对象，或设定对象的初始值，或复制对象等。而对象函数的主要用途是存取对象里的数据值。

以上的 **machine.h** 文件只对对象函数和类函数做定义而已，至于函数的实现内容 **(Implemetation)**则从缺。于是还需要把函数的实现内容编写在下述的 **machine.c** 文件中。

```
----范例 ex1-01(二)----
/*  machine.c  */
#include <stdio.h>
#define IMPLEMENTATION
```

```
#include <machine.h>
/* Object implementation */
float
constMethodDecl(computeSalePrice)
{  return (this->m.price)* 0.8;  }
float
constMethodDecl(inquireHeight)
{  return this->m.height;  }
void
constMethodDecl(printName)
{  printf("品牌:\t%s\n", this->m.name);  }
BASEOBJECT_IMPLEMENTATION
    methodName(computeSalePrice),
  methodName(inquireHeight),
  methodName(printName)
ENDOF_IMPLEMENTATION
/*  Class implementation  */
initClassDecl(){}
dtorDecl()
{ free((void*)this->m.name);
  this->m.name = NULL;
  free((void*)this->m.year);
  this->m.year = NULL;
}
t_machine
classMethodDecl_(*const new)char const name[], char const year[],
float height, float price __
{ t_machine *const this = machine.alloc();
```

```
  if (this)machine.init(this, name, year, height, price);
  return this;
}
void
methodDecl_(init)char const name[], char const year[],
    float height, float price __
{ this->m.name = strdup(name);
  this->m.year = strdup(year);
  this->m.height = height;
  this->m.price = price;
}
void
methodDecl_(copy)t_machine const*const pobj __
{ machine._machine(this);
  machine.init(this, pobj->m.name, pobj->m.year, pobj->m.height,
  pobj->m.price);
}
CLASS_IMPLEMENTATION
  methodName(new),
  methodName(init),
  methodName(copy)
ENDOF_IMPLEMENTATION
```

其中的指令**/* Object implementation */**部分就是对象函数的实现内容了。而指令**/* Class implementation */**部分则是类函数的实现内容了。

至此，已经定义好了 **machine** 类了。于是可以编写 **main()**函数来定义 **machine** 对象，并传递消息给它。

----范例 ex1-01(三)----

```
/*   xmain.c   */
#include <stdio.h>
#include <machine.h>
int main(void)
{        float sale_price, height;
      t_machine *a = machine.new("夏普", "2006", 0.83, 5000.0);
         /* send messages */
    sendMsg(a, printName);
    sale_price = sendMsg(a, computeSalePrice);
    height = sendMsg(a, inquireHeight);
    printf("height : %.2f\n", height);
    printf("sale_price : %.2f\n", sale_price);
    /* delete object */
    delete(a);
    getch();
    return EXIT_SUCCESS;
}
```

此程序输出：

```
品牌:  夏普
height : 0.83
sale_price : 4000.00
```

类函数的目的并非在于处理对象的内容，而是：

- 诞生对象，设定对象初值，复制或比较对象等。
- 处理些有关于整个类的事情。例如累计总共诞生多少个对象、或把刚诞生的对象存入 Linked List 里等管理众多对象的事情。

调用类函数的格式为：

```
类. 类函数
```

例如：

```
machine.new("夏普", "2006", 0.83, 5000.0)
```

这指令调用 **new()**类函数来诞生对象，并且传入数据以设定对象的初始值。您也可以解释为 **main()**主函数把 **new("夏普", "2006", 0.83, 5000.0)**消息传递给 **machine** 类。

有时候，一般的对象函数是用来处理对象内的数据，所以调用对象函数的格式为：

```
sendMsg( 对象指针, 对象函数 )
```

例如，

```
sendMsg(a, printName)
```

这指令调用 **printName()**对象函数来存取对象里的数据内容。您也可以解释为 **main()**主函数把 **printName** 消息传递给 **a** 对象。再来看看上述主函数中，对象 **a** 首先接到一个消息 **computeSalePrice**，对象 **a** 就会启动其内部的 **computeSalePrice()**函数，此函数就拿 **0.8** 和对象中的价格 **5000** 相乘，函数计算完毕得到 **4000**，对象 **a** 就将其送出来。

接下来，对象 **a** 又接到另一消息 **inquireHeight**，对象 **a** 启动其内部的 **inquireHeight()**函数，此函数就读取对象内的“高度”**0.83**；且把 **0.83** 送出来。最后，电脑把高度及售价金额显示在屏幕上。

1.13.4 对象的分类

OOP 的观念能大量简化电脑软件的复杂度，让人们更容易发展及维护软件。**OOP** 的两个最基本观念是：类**(Class)**及对象**(Object)**。俗语常说：“物与类聚”，

意味着类似之物常常聚在一堆，这一堆就是“类”，而其组成元素就是“对象”了。根据传统的程序写法，主要的程序设计工作在于设计指令、描述及函数等；然而在 **OOP** 的新观念中，编写程序的主要工作在于设计类。设计各式各样的类后，就能使用类的对象，而产生有价值的信息。请看如何设计一个名为叫 **Tree(**树**)** 的类，其格式为：

```
类 Tree
{
    品种
        年龄
    高度
}
```

对于 **Tree** 类的树**(**对象**)**，将记录其 **3** 种属性**(Attribute)**：品种、年龄及高度。例如有一棵树，其属性值为：

```
品种：peach
年龄：8 年
高度：2.1m
```

这是树种中的一棵树，在电脑的 **Tree** 类中，就必须有一个“对象”**(Object)**和它对应并且储存它的特性**(**征**)**。至于如何产生 **Tree** 类的对象呢？其 **OOPC** 指令为：

```
t_Tree *x = Tree.new();
```

此时电脑就定义对象 **x**，**x** 能储存此树的数据，如图 **1-41** 所示。

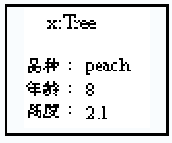


图 1-41　定义对象 x

如果您想记录别的树的数据，可再定义 **Tree** 类的对象如下：

```
t_Tree *y = Tree.new();
```

对象 **y** 就可储存此树的数据了。实物上的关系：

```
树林 <——> 树
```

其表现在电脑程序上是：

```
类 <——> 对象
```

此为 **OOP** 的基本观念。类设计的优劣常决定软件的生命和价值，好的设计其软件寿命源远流长，价值连城。类的设计并非到此为止，还要继续做一项重要工作，即分类**(Classification)**，即寻找“子类”**(Subclass)**。例如，上述树若可分为果树与竹子两种，则 **Tree** 类可分为“果树”**(Fruit_tree)**与“竹子”**(Bamboo)**两个子类。

此为实物上的类关系。其中，**Fruit_tree** 及 **Bamboo** 是 **Tree** 的子类**(Subclass)**，而 **Tree** 是 **Fruit_tree** 及 **Bamboo** 的超类**(Superclass)**。**Fruit_tree** 类中含有“成熟月份”及“价格”两项新特性。这意味着，对于果树，必须多储存两项数据。这两项数据是果树才有的，竹子就没有。**Bamboo** 类中含有一项新特性，这告诉电脑说：对于竹子，必须多储存一项数据即“用途”。这是竹子才有的数据，果树则无。例如有一棵果树，其数据为：

```
品种：peach
年龄：8 年
高度：2.1 m
成熟月份：3 月
价格：20 元
```

在 **OOP** 程序中，必须定义 **Fruit_tree** 类的对象来储存这些数据。欲产生此对象，**OOPC** 程序可写为：

```
t_Fruit_tree *a = Fruit_tree.new();
```

此时电脑就产生一个 **Fruit_tree** 类的对象，叫 **a**。它包含 **5** 项数据，如图 **1-42** 所示。

图 1-42　Fruit_tree 类的对象 a

其中包含了 **Tree** 类内的 **3** 项数据以及 **Fruit_tree** 类专有的两项数据。因为 **Tree** 为 **Fruit_tree** 的超类，所以 **Fruit_tree**“继承”**(Inherit)**其超类 **Tree** 的 **3** 个特性，如图 **1-43** 所示。

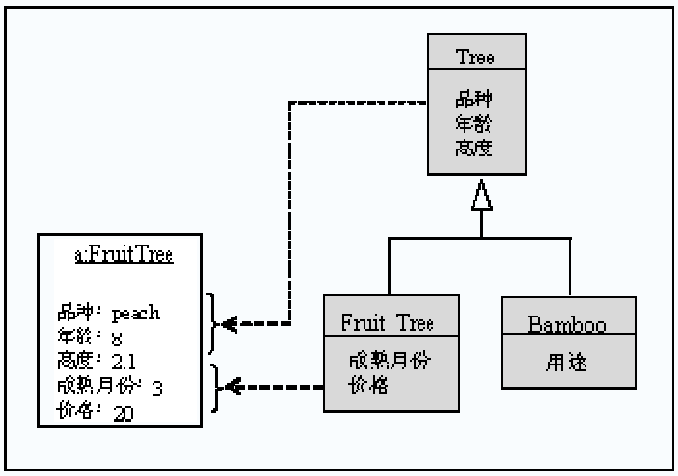


图 1-43　数据的继承

超类 **Tree** 内的数据为子类 **Fruit_tree** 和 **Bamboo** 都具有的共同特性，此为 **OOP** 类设计的基本原则。类之间有些互为独立，有些具有密切的“父子”关系；由于儿女常继承父母的生理或心理特征，所以又称此关系为“继承”**(Inheritance)** 关系。类间的密切关系，把相关的类组织起来，并且组织程序内的对象。若程序内的对象毫无组织，呈现一片散沙，就不是好程序。完美的 **OOP** 程序，必须重视类间的关系，对象是有组织的。

1.13.5　定义继承关系

如果 **A** 类“继承”**B** 类，则称 **A** 为“子类”**(Subclass)**，也称 **B** 为“超类”**(Superclass)**，即 **B** 为 **A** 的超类，**A** 为 **B** 的子类。也许您觉得“继承”的观念很陌生，不知如何看出类间的继承关系。下列两个描述的意义相同。

(1) A 为 B 的子类。

(2) A 为 B 的一种**(a kind of)**特殊类。

根据描述**(2)**能轻易找到父子关系。例如，威尔森**(Wison)**生产高质量球拍，球拍分网球拍与羽球拍两种。从此句子得知，网球拍为一种**(a kind of)**球拍，羽球拍也为一种球拍。因此，网球拍为球拍的子类，羽球拍也为球拍的子类，即球拍是超类，如图 **1-44** 所示。

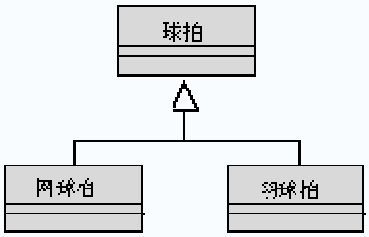


图 1-44　超类与子类

程序不仅要定义类，也要定义其继承关系。前面已经介绍如何定义类以及对象了，本节将告诉您如何定义类的继承关系，在此举例说明，如图 **1-45** 所示。

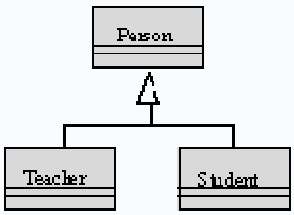


图 1-45　定义类的继承关系的方法

程序的设计过程如下。

(1) 定义超类。例如定义 **Person** 类，如下所示。

```
---范例 ex1-02(一)---
/*    person.h    */
#ifndef  PERSON_H
#define  PERSON_H
#include  <ooc.h>
#undef   OBJECT
```

```
#define OBJECT Person
/* Object interface */
BASEOBJECT_INTERFACE
  char *private(name);
  int private(age);
BASEOBJECT_METHODS
 void constMethod(Display);
 void constMethod(YearOfBirth);
ENDOF_INTERFACE
/* Class interface */
CLASS_INTERFACE
  t_Person *const classMethod_(new)char const name[], int age __;
  void method_(init)char const name[], int age __;
ENDOF_INTERFACE
#endif
```

---范例 ex1-02(二)---

```
/*   person.c   */
#include <stdio.h>
#define IMPLEMENTATION
#include <person.h>
/*  Object implementation */
void constMethodDecl(Display)
{   printf("Name: %s, Age: %d\n", this->m.name, this->m.age); }
void constMethodDecl(YearOfBirth)
{   printf("YearOfBirth: %d\n", 2006 - this->m.age);  }
BASEOBJECT_IMPLEMENTATION
 methodName(Display),
```

```
 methodName(YearOfBirth)
ENDOF_IMPLEMENTATION
/*  Class implementation  */
initClassDecl()
{} /* required */
dtorDecl()/* required */
{ free((void*)this->m.name);
  this->m.name = NULL;
}
t_Person classMethodDecl_(*const new)char const name[], int age __
{ t_Person *const this = Person.alloc();
  if (this)Person.init(this, name, age);
  return this;
}
void methodDecl_(init)char const name[], int age __
{ this->m.name = strdup(name);
  this->m.age = age;
}
CLASS_IMPLEMENTATION
  methodName(new),
  methodName(init)
ENDOF_IMPLEMENTATION
```

(2) 定义子类。例如定义 **Teacher** 类，如下所示。

```
---范例 ex1-02(三)---
/*  teacher.h  */
#ifndef TEACHER_H
#define TEACHER_H
```

```
#include <person.h>
#undef  OBJECT
#define OBJECT Teacher
/* Object interface */
OBJECT_INTERFACE
  INHERIT_MEMBERS_OF(Person);
OBJECT_METHODS
  INHERIT_METHODS_OF (Person);
ENDOF_INTERFACE
/* Class interface */
CLASS_INTERFACE
  t_Person * const classMethod_(new)char const name[], int age __;
  void method_(init)char const name[], int age __;
ENDOF_INTERFACE
#endif
```

---范例 ex1-02(四)---

```
/*   teacher.c    */
#include <stdio.h>
#define IMPLEMENTATION
#include <teacher.h>
/*   Object implementation   */
OBJECT_IMPLEMENTATION
   SUPERCLASS(Person)
ENDOF_IMPLEMENTATION
/*  Class implementation   */
initClassDecl()/* required */
{  initSuper(Person); }
```

```
dtorDecl()/* required */
{  Person._Person( super(this,Person)); }
t_Teacher classMethodDecl_(*const new)char const name[], int age __
{  t_Teacher *const this = Teacher.alloc();
    if (this)Teacher.init(this, name, age);
    return this;
}
void methodDecl_(init)char const name[], int age __
{  Person.init( super(this,Person), name, age);  }
CLASS_IMPLEMENTATION
  methodName(new),
  methodName(init)
ENDOF_IMPLEMENTATION
```

目前已经定义好 **Person** 和 **Teacher** 两个类，其中的指令如下：

```
OBJECT_INTERFACE
  INHERIT_MEMBERS_OF(Person);
OBJECT_METHODS
  INHERIT_METHODS_OF (Person);
ENDOF_INTERFACE
```

表达了 **Teacher** 为 **Person** 的子类。

现在，已知道如何表达继承关系了，然而，子类从超类继承什么东西呢？类包含“数据成员”及“方法”。因此从上述指令可以看出，子类继承超类的数据成员**(Data Member)**及方法**(Method)**。

所谓继承数据成员，表示继承数据成员的定义，而不是继承数据成员的值，请详加区别。类定义数据成员**(**含类型及名称**)**，对象诞生后，对象内才有数据值。所以“类继承”是继承类的定义，不是继承对象的值。也就是说，若超类定义 **nam**

e 及 age 两个数据成员，则子类天生就拥有此两个数据成员，所以子类不需定义它们。所谓继承方法，表示子类天生就拥有超类定义的方法。

例如，Person 的子类天生就具有 YearOfBirth()及 Display()方法。于是可以通过 Teacher 对象而存取到 Person 类里的数据成员。

接着，请看如下主程序。

```
---范例 ex1-02(五)---
/*   xmain.c   */
#include <stdio.h>
#include <teacher.h>
int main(void)
{
  t_Teacher *t = Teacher.new("Mike", 20, 76500.00);
  sendMsg(super(t, Person), Display);
  sendMsg(super(t, Person), YearOfBirth);
  delete(t);
  getch();
  return EXIT_SUCCESS;
}
```

此程序输出：

```
Name: Mike, Age: 20
YearOfBirth: 1986
```

表面上 Teacher 类中，未定义数据成员及方法，但事实上已拥有 name 及 age 两个数据成员，也拥有 YearOfBirth()及 Display()两个方法。因此，Teacher 类的对象都能调用 YearOfBirth()及 Display()方法。

在应用上，子类也常拥有自己的数据成员及方法，使其有别于超类。例如将上述 Teacher 类改写如下所示。

---范例 ex1-03(一)---

```
/*   person.h
   的内容与前面范例 ex1-02 相同。*/
```

---范例 ex1-03(二)---

```
/*   person.c
   的内容与前面范例 ex1-02 相同。*/
```

---范例 ex1-03(三)---

```
/*   teacher.h   */
#ifndef  TEACHER_H
#define  TEACHER_H
#include  <person.h>
#undef   OBJECT
#define  OBJECT  Teacher
/*  Object  interface  */
OBJECT_INTERFACE
  INHERIT_MEMBERS_OF(Person);
  double  private(salary);
OBJECT_METHODS
 INHERIT_METHODS_OF  (Person);
 void  constMethod(Show);
 void  constMethod(Show_2);
 void  constMethod(Show_3);
ENDOF_INTERFACE
/*  Class  interface  */
CLASS_INTERFACE
  t_Person  *  const  classMethod_(new)
     char  const  name[],
```

```
    int age,
    double salary __;
  void method_(init)
    char const name[],
    int age,
    double salary __;
ENDOF_INTERFACE
#endif
```

---范例 ex1-03(四)---

```
/*   teacher.c   */
#include <stdio.h>
#define IMPLEMENTATION
#include <teacher.h>
/*   Object implementation   */
void
constMethodDecl(Show)
{       printf("Name: %s, Age: %d\n",
      this->m.Person.m.name,
      this->m.Person.m.age);
  printf("Salary: %.2f\n", this->m.salary);
}
void constMethodDecl(Show_2)
{       printf("Name: %s, Age: %d\n",
      super(this, Person)->m.name,
       super(this, Person)->m.age);
  printf("Salary: %.2f\n", this->m.salary);
}
```

```
void constMethodDecl(Show_3)
{       sendMsg(super(this, Person), Display);
  printf("Salary: %.2f\n", this->m.salary);
}
OBJECT_IMPLEMENTATION
   SUPERCLASS(Person),
   methodName(Show),
   methodName(Show_2),
   methodName(Show_3)
ENDOF_IMPLEMENTATION
/* Class implementation */
initClassDecl()/* required */
{  initSuper(Person);
}
dtorDecl()/* required */
{ Person._Person( super(this,Person));
}
t_Teacher classMethodDecl_(*const new)
     char const name[], int age, double salary __
{
    t_Teacher *const this = Teacher.alloc();
    if (this)Teacher.init(this, name, age, salary);
    return this;
}
void methodDecl_(init)char const name[], int age, double salary __
{
   Person.init( super(this,Person), name, age);
   this->m.salary = salary;
```

```
}
CLASS_IMPLEMENTATION
  methodName(new),
  methodName(init)
ENDOF_IMPLEMENTATION
```

现在，**Teacher** 类含有 **3** 个数据成员。

- name：从 Person 类继承而来。
- age：从 Person 类继承而来。
- salary：自定义的。

此外，也含有 **5** 个方法。

- YearOfBirth()：从 Person 继承而来。
- Display()：从 Person 继承而来。
- Show()：自定义的。
- Show_2()：自定义的。
- Show_3()：自定义的。

由于 **Teacher** 从 **Person** 继承而得到 **Person** 类所定义的 **init()**方法，也算是 **Teacher** 的方法了，所以 **Teacher** 自定义的 **init()**能调用 **Person** 的 **init()**来设定 **name** 及 **age** 值；之后，**Teacher** 的 **init()**方法设自己的 **salary** 值。同样地，**Show_3()**方法也能调用 **Person** 的 **Display()**来显示 **name** 及 **age** 的内容；之后，**Show_3** 自己输出 **salary** 的值。请看个主程序。

```
---范例 ex1-03(五)---
/*   xmain.c   */
#include  <stdio.h>
#include  <teacher.h>
int  main(void)
{
  t_Teacher  *t1  =  Teacher.new("Mike",  20,  76500.00);
```

```
    t_Teacher *t2 = Teacher.new("Lily", 22, 86000.00);
    sendMsg(t1, Show);
    printf("\n\n");
    sendMsg(t1, Show_2);


    printf("\n\n");
    sendMsg(t2, Show_3);
    delete(t1);
    delete(t2);
    getch();
    return EXIT_SUCCESS;
}
```

此程序输出如下：

```
Name: Mike, Age: 20
Salary: 76500.00
Name: Mike, Age: 20
Salary: 76500.00
Name: Lily, Age: 22
Salary: 86000.00
```