第一天　C++历史和特征

今天将概括地介绍C++的一些关键概念。C++作为面向对象程序设计语言，与其面向对象的特征是有密切关系的。在许多地方讨论C++的面向对象一个的特征时，由于这些特征的相互关联，所以还需要其他特征的知识。为讨论这些问题，本章概述了大部分C++中重要的特征和概念。以后的几天将详尽地学习C++的这些特征。

1.1　C++的起源

众所周知，C++是C的扩充版。C++对C的扩充首先是由新泽西州Murray Hill贝尔实验室的Bjarne Stroustrup在1980年提出的，他称之为"含类的C"，后来到1983年更名为C++。

虽然C++的前身C在世界上是一种人们非常喜欢并被广泛使用的专业编程语言，但C程序的复杂性使C++的出现成为必然。在C中如果一个程序代码达到25000至100000行，那它将变的过于复杂使人很难从整体上把握它。C++解决了这种困难，C++的实质是使程序员易于理解和管理更大更复杂的程序。

Stroustrup对C作的大部分扩充是支持面向对象程序设计的，有时被看作面向对象程序设计(详见下节对面向对象程序设计的简要解释)。Stroustrup声称C++的一部分面向对象的特征来自于另一种面向对象程序设计语言Simula67。因此，C++成为两种强有力的程序设计方法的结合体。

因为C++的可取之处，使它度过了两次大的波折，一次在1985年，另一次在1989年。当前版本为2.1，即是本书讨论的这一版。

在C++提出时，Stroustrup就意识到保持C的优点的重要性，包括C的效率以及C的灵活性和程序员而不是语言作为主管的潜在道理，同时提供对面向对象程序设计的支持。他的目标得到了实现，C++使程序员依然能自由灵活地控制C，并充分发挥对象的作用。C++的面向对象的特征，用stroustrup的话来说，"使程序更清晰、便于扩展，在不降低效率下使维护程序更方便。"

尽管C++初始设计目标是为更好的管理大型程序，但这并没有限制其他方面的用处。实际上，C++的面向对象的特性使C++可以有效地应用于任何程序设计的任务，C++用于设计诸如编辑器、数据库、个人文件系统及通讯程序并非希罕的事情。而且，由于C++拥有C的效率，已有许多高效系统软件由C++设计。

1.2　什么是面向对象程序设计

面向对象程序设计(OOP)是一种完成程序设计工作的新方法。自从计算机发明以来，程序设计方法发生了戏剧性的变化，这些变化主要是为了适应处理日益增长的程序的复杂

机。比如，在计算机刚出现时，程序设计是通过利用计算机前控制板输入二进制机器指令来实现。当程序只有几百条指令时，这种方法可以胜任。随着程序的增长出现了汇编语言，使程序员可以用符号化的计算机指令处理更大更复杂程序。随着程序的进一步的增长，出现了高级语言。它为程序员处理复杂程序提供了更好的方法。第一个广泛流行的高级语言当属FORTRAN，虽然FORTRAN的出现是富于开创性的一步，但用它很难开发出清晰易懂的程序。

1969以后出现了结构化程序设计方法，这就是诸如C和PASCAL等语言支持的方法。使用结构化程序设计语言建编写中等复杂性的程序相当容易。然而，当任务达到一定大小时，使用结构化程序设计方法也会变得失去控制。它的复杂度超过程序员所能管理的程度和结构化方法所能胜任的极限。

试想，程序设计发展的每一个里程碑，都有新一方法被采用去处理越来越复杂的程序，前进的每一步，新的方法都吸取了旧方法的精华并继续向前发展。现在，许多工程都接近或正在结构化编程不起作用的节点上。为解决这个问题，人们采用了面向对象程序设计的方法。

面向对象程序设计方法吸取了结构化程序设计方法的先进思想并结合了许多有效的新概念，使程序员用一种新途径完成程序的设计。总体上来说，采用面向对象的风格进行程序设计时，程序员将问题分解为相互联系部分的了组。每个部分包括同其它子组联系的数据和代码。而且可以将这些子集组织成有层次的结构。最后，可以将这些子组转换成自约束的单元，称为对象。

所有面向对象程序设计语言都含有三个概念：对象、多态性、继承。

1.2.1 对象

面向对象程序设计语言最重要的特征是对象。简单地说，一个对象是一个逻辑实体，包含数据及处理数据的代码。每对象体中，一些数据和代码是属于对象私有的，不允许对象外的实体访问。因此，对象提供了一种标识表示其他实体的保护等级，使其不能偶然修改或不正确的使用对象中的私有部分。这种防止公共使用代码的方法称为封装。

对于所有需求，一个对象是用户定义类型的一个变量，开始对于将这个抽象型和代码联结在一起的对象看做是一个变量，好像程序在转，然而在面向对象程序设计中的确如此。当程序员定义了一个对象时，无疑地创建了一个新的数据类型。

1.2.2 多态性

面向对象程序设计的语言支持多态性，多态性的特性可以用"一个界面，多个方法"来描述。简单地说就是一个名称可以完成几个相关但略有不同功能。实质上，多态性允许一个界面用来进行一类行为。具体的行为由其所包括的数据类型来确定。例如，有一个程序定义了一种类型的堆栈。一个是处理整数值的，一个是处理浮点数的，一个是处理长整数值的。因为多态性，你可以为PUSH(I)和PUSH(D)创立一种类型的函数。一种类型的数据一种函数，通过对调用PUSH(I)和PUSH(D)的数据类型的识别，编译器可以选择正确的路径。整体的概念(界面)的意思是将数据弹出和压入堆栈，函数定义的具体方法是为了处理不同的数据类型。

第一个面向对象程序设计语言就解释执行的，所以对多态性的支持是运行时实现的。而

C++是编译执行的，所以C++在运行和编译时都提供对多态性的支持。

1.2.3 继承

继承是指对一个对象可以接收另一个对象的属性的处理。这很重要，因为它支持类的概念，许多知识因为类的继承而易于管理。例如，一个美味的红苹果是苹果类的一部分，而苹果则是属于水果类的，水果又同样属于更大的食物类。不使用类的概念，每个对象在定义时要明确地列出所有的特性。然而，通过类的使用，在定义对象时只需列出有别于类中其他对象的那些特性即可。继承机制使对象作为更普通的例子中具体的例子。

1.3 C++的编程风格

因为C++是C的一个超集，C++的程序可以写的像C的程序。然而，这样将限制充分发挥C++的长处(就像在看彩电时将彩色关闭)，然而多数程序员使用对C++是唯一的风格和特征。C和C++的风格的差异主要是因为充分发挥C++的面向对象的能力而造成的。使用C++的编程风格的另一个优点是帮你开始用C++的方式思考而不再用C(就是说，当写C++代码时，通过接受另一种不同的风格，使你停止用C而开始用C++的方式思考)。

既然学会将C++的程序写的像C++程序是非常重要的，这一节将介绍一些这方面的特征。看一下这个C++程序：

```
#include "iostream.h"

main()
{
  int i;

  cout << "这是输出.\n";  // 这是单行注释
  /* 也可以使用C的注释语法 */

  // 用>>输入一个数
  cout << "输入一个数:";
  cin >> i;

  // 用<<输出一个数
  cout << i << "平方是" << i*i << "\n";

  return 0;
}
```

的确，这个程序同一般的C程序有很大的不同。首先，需要包含iostream.h头文件。这个头文件是由C++定义的，用于支持C++风格的I/O操作(因为C++比较新，有些编译器对标准头文件采用略微不同的名字)。

第一个风格的差异体现在这一行：

```
cout << "这是输出.\n";  //这是单行注释
```

这一行体现了C++的两个特征。首先,语句

```
cout << "这是输出.\n";
```

将"这是输出."显示到屏幕上紧接着回车换行。在C++中,<<是一个扩展符号,它仍然是左移运算符,但当它像上例中这样使用时,它就成为输出操作符了。cout是同屏幕连接的标识符(实际上像C一样,C++也支持I/O的重定向,但为了讨论方便起见,假设cout是指向屏幕的)。在C++中,可以用cout和<<输出任何一种固有的数据类型,以及字符串。

在C++程序中,仍然可以使用PRINTF( )和C的I/O函数(在使用这些函数时必须包含stdio.h)。然而,多数程序员认为使用cout<<更能体现C++的精神。进一步来说,在本例中使用<<输出字符串与使用PRINTF( )实际上是一样的,而C++的I/O系统可以自动扩展执行对象中定义的操作(在很多情况下,printf( )却不能完成)。

总的来说,C++程序可以使用ANSI C所支持的任何标准库函数。然而,在许多地方如果C++提供了新的方法,像<<运算符,本书将使用新的方法来替代C的标准库函数(虽然不存在强制的规则)。

输出表达式之后是C++的注释行。C++中,注释行可以用两种方式定义,第一种是使用C的注释方法,这在C++中同样起作用。而在C++中,也可以用//定义一个单行注释。当使用//时,编译器将忽略从//到该行行尾的所有字符。一般说来,C++的程序员在定义多个注释行时用C的定义方法,而在只需要单个注释行时使用C++的单行注释定义法。

接着,程序提示用户输入一个数字。数字通过以下语句由键盘读入的:

```
cin >> i;
```

在C++中,>>运算符仍然有右移的功能。然而,当它像上述一样使用时,>>又成了C++的输入运算符。这个语句从键盘为变量i赋值。标识符cin指明足从键盘获值。一般地说,可以使用cin >>为任何基本数据类型变量及字符串赋值。

注:这行代码并非印错。具体来说,i的前面并不需要&。当然,当使用SCANF()输入信息时,必须传递一个指向接受信息的变量的指针。这就是说,要在变量名前加上取地址操作符&。然而,>>运算符是C++的扩充,所以不需要(实际上是必须不)&。在第三天"数组、指针和引用"将解释为什么。

在上例中,也可以使用任何C的输入函数,像使用SCANF()替代cin >>。当然像cout一样,大多数程序员认为cin >>更能体现C++的精神。

另一个令人感兴趣的一行是:

```
cout << i << "平方是" << i * i << "\n";
```

如上所述,这个语句在屏幕上输出"10平方是100"(假设i=10)及回车换行。像上行所描述,可以同时使用几个<<输出运算符。

如上所述,在输入输出操作中,<<和>>运算符能处理C++的任何固有数据类型及字符串。例如,下面的程序将输入一个浮点数、一个双精度数和一个字符串然后输出到屏幕上。

```
#include "iostream.h"

main()
```

```
{
  float f;
  char str[80];
  double d;

  cout << "输入两个浮点数: ";
  cin >> f >> d;

  cout << "输入一个字符串: ";
  cin >> str;

  cout << f << " " << d << " " << str;

  return 0;
}
```

运行这个程序,在提示输入字符串时键入"This is a string."当程序重新显示输入的数据时,字符串中只有"This"被显示出来。这是因为>>运算符的工作方式同带%s参数的SCANF()函数是一样的,当它遇到第一个空格符时就停止了读入。所以程序没有读入"is a string"。

这个程序也说明可以将几个输入运算符连在一起使用。

在写C和C++程序时的另一个差异是说明局部变量的位置。用C,说明局部变量要集中在块的开始部分,却不能在语句出现后再进行变量说明,以下这一段程序就是不正确的:

```
/* 在C中不正确 */
f()
{
  int i;

  for(i=0; i<10; i++) {
    int j; /* 作为C程序,不能编译 */
    j = i * 2;
    .
    .
    .
  }
}
```

因为for循环被变量i和j的说明所隔断,C编译器将拒绝编译并给出错误标识。而在C++中,这段程序将很容易被接受而不会出现编译错误。因为C++允许在块内任何位置进行变量说明,而不是一定要集中在块首。

下面是前面程序的另一个版本,每一个变量在需要的地方被说明。

```
#include "iostream.h"

main()
{
```

```
float f;
double d;
cout << "输入两个浮点数:";
cin >> f >> d;

cout << "输入一个字符串:";
char str[80]; // str 在此声明,在第一次引用的前面
cin >> str;

cout << f << " " << d << " " << str;

return 0;
}
```

在块首说明所有的变量或在第一次使用时进行说明都由编程者决定。既然数据和代码的封装是C++的风格,因此将变量的说明放在使用它们的地方而不是在块首仍然是合理的。上面的程序中,将变量的说明分开仅仅是为了说明。而且很容易想像,C++的这些特征在更为复杂的例子中会更有价值。

在需要的地方说明变量可以避免偶然的副作用。然而,在大的函数中这种说明方法会带来更大的好处。坦率的说,在小程序中(像本书中的例子),没有理由不可以在程序开始处说明变量。所以本书只在比较大且复杂的函数中,而这种方式可能有用时,才使用这种即用即说明的方式来说明变量。

对局部化变量说明方式的普遍优势也有一些不同的意见。他们认为块内分散的变量说明使人在阅读代码时迅速找到块中所有的变量变的更困难,使程序维护更困难。因此,许多C++程序员并不重视这个特征的使用。在这个问题上,本书并不坚持任何一种方法。然而对于大的函数中,如果使用的合适,在第一次使用时说明变量有助于编写出高质量的程序。

1.3 C++的类

本节介绍C++最要的特征:类。在C++中要想创建一个对象,首先要使用关键词CLASS定义它的一般形式。类在语法上与结构相似,下面定义了一个叫做堆栈的类,它可以用来创建一个堆栈:

```
#define SIZE 100

// 定义一个创建类 stack
class stack {
  int stack[SIZE];
  int tos;
public:
  void init();
  void push(int i);
```

```
  int pop(void);
}
```

类可以包含私有和公有部分。缺省条件下，类定义的所有属性被认为是私有的，例如，上例中变量 stck 和 tos 是私有的。这就是说它们不能被不属于此类的函数所使用。通过定义指定数据项为私有来控制对它们的访问，是实现封装的一个途径。虽然上例中没有定义私有函数但也可以定义函数为私有，使它们只能被本类中的成员所调用。

要定义类中的公有部分(即它们可以被程序中其他部分所访问)，必须在它们的说明之前加上关键字 public，所有在关键字 public 后定义的变量和函数都可以被程序中其他函数所访问和调用。实际上，程序的其他部分就是通过对象的公有函数来进行对象访问的。需要提到的是，虽然可以使用公有变量，但最好限制它们的使用，最好将所有数据定义为私有，而通过对象的公有函数来访问它们。另外一点是，关键字 public 后要加冒号。

函数 init( )，pop( )，push( )叫做成员函数，因为它们是属于堆栈类的。记住，对象使代码和数据连接在一起，只有本类成员函数才能访问本类中定义的私有变量。

如果你定义了一个类，可以用类名来创建这种类型的对象。实际上，类名就成为新的数据类型的标识符。例如，定义一个叫做 mystack 的堆栈类型的对象：

```
stack  mystack;
```

也可以在定义类的花括号后定义这种类型的变量(对象)，这点和结构类型是一样的。

如上所述，在 C++中一个类创建了一种新的类型，可以使用它创建该类型的对象。因此，如同一些变量是整型数实例一样，对象也是类的一个实例。换句话说，类是一个逻辑抽象，而对象是一个实体(也就是说，对象存在于计算机的内存中)。

类声明的一般形式：

```
class 类名{
私有数据和函数
  public:
公用数据函数
}对象名表;
```

当然，对象名表可以是空的。

在 stack 类的声明中，使用了成员函数的原型。在 C++中理解这一点是很重要的，当告诉编译器关于一个函数时，必须给出它的完整的原型。(实际上，在 C++中必须给出函数的原型，这上点并非可选的。)

当为一个类的一个成员函数编写代码时，必须告诉编译器该函数所属的类名，这可以通过在函数名前加上该函数所属类的名来完成，下面的例子是编写函数 push( )的一种方法：

```
void stack::push(int i)
{
  if(tos==SIZE) {
    cout << "堆栈已满";
    return;
  }
  stck[tos] = i;
  tos++;
```

}

称做作用域限定符。本质上，它告诉编译器版本的函数属于 stack 类，或者换句话说，push()函数是在 stack 类范围内的。如上所见，在C++中，几个不同的类可以使用同名的函数。通过作用域限定符和类名的使用，编译器可以确知每一个函数数据所属的类。

在程序中不属于某类的部分调用该类的成员函数，必须使用对象名加上点运算符以及该函数的名称。例如，下面是对对象 stack1 的 intit()函数的调用：

```
stack stack1,stack2;
stack1.init();
```

在这里，一定要理解 stack1 和 stack2 是两个不同的对象。这就是说，对于对象 stack 的初始化不会影响到对象 stack2。对象 stack1 和 stack2 的唯一关系是它们是同一种类型。

一个成员函数可以直接调用另一个成员函数，而不需要使用点运算符。只有成员函数被不属于该类的代码调用时才必须使用点运行符和变量名。

下面的例子将上面关于 stack 类的片断集成起来并添加了一些细节，该例子完整的描述了 stack 类：

```
#include "iostream.h"

#define SIZE 100

// 下面创建类 stack
class stack {
  int stck[SIZE];
  int tos;
public:
  void init();
  void push(int i);
  int pop(void);
};

void stack::init()
{
  tos = 0;
}

void stack::push(int i)
{
  if(tos==SIZE) {
    cout << "堆栈已满";
    return;
  }
  stck[tos] = i;
  tos++;
```

```
}

int stack::pop()
{
  if(tos == 0) {
    cout << "堆栈下溢";
    return 0;
  }
  tos--;
  return stck[tos];
}

main()
{
  stack stack1,stack2;  // 创建两个堆栈对象

  stack1.init();
  stack2.init();

  stack1.push(1);
  stack2.push(2);

  stack1.push(3);
  stack2.push(4);

  cout << stack1.pop() << " ";
  cout << stack1.pop() << " ";
  cout << stack2.pop() << " ";
  cout << stack2.pop() << "\n";

  return 0;
}
```

切记:一个对象的私有部分只能被该对象的成员函数所访问。例如下面的语句

```
stack1.tos=0;//error
```

不能在上例的 main()函数中出现,因为成员 tos 是私有的。

习惯上,大多数的 C 程序将 main()函数作为程序的第一个函数。然而,在 stack 程序中,stack 类的成员函数在 main()函数之前被定义。尽管没有规定表示必须这样作(它们可以在程序的任何地方定义),但这是编写 C++程序时最常见的方式(然而,非成员函数仍然在 main()函数后定义),本书将服从这种惯例。当然,在实际应用中,同程序相关的类往往包含在一个头文件中。

1.5 函数重载

C++实现多态性的一种方式是通过使用函数的重载。在C++中如果两个或两个以上的函数有不同的参数声明则它们可以共享同一个名字。在这种情况下几个函数共享一个名字称为重载，这个处理过程称为函数的重载。

为什么函数的重载如此重要？首先看下面三个在所有的C编译器都可以找到的标准库函数：abs()，labs()和fabs()。函数abs()返回一个整数的绝对值，函数labs()返回一个长整数的绝对值，而函数fabs()返回一个双精度数的绝对值。尽管这些函数执行的是本质上相同的工作，但在C中必须使用三个不同的名字来表示它们。尽管每个函数的潜在含义是相同的，而程序员必须记住三个而不是一个名字。然而，在C++中可以对这二个函数使用同一个名字，如下面程序描述的：

```
#include "iostream.h"

// abs 被以二种方式重载
int abs(int i);
double abs(double d);
long abs(long l);

main()
{
  cout << abs(-10) << "\n";

  cout << abs(-1.0) << "\n";

  cout << abs(-9L) << "\n";

  return 0;
}

int abs(int i)
{
  cout << "使用整数 abs()\n";

  return i<0 ? -i : i;
}

double abs(long d)
{
  cout << "使用双精度 abs()\n";

  return d<0.0 ? -d : d;
```

```
}

long abs(long l)
{
  cout <<"使用长整数 abs()\n";

  return l<0 ? -l : l;
}
```

该程序建立了三个相似但不同的abs()函数，每一个函数返回它的参数的绝对值。因为每一个函数的参数的不同，编译器在每次调用时都能确认被调用的函数。重载函数的价值是它允许一系列相似的函数可以按同一个名字被访问。因此，名字abs()就表示了它所执行的一般性的行为。对于在一个具体的环境中选择一个相应的版本就是编译器的事情了，程序员只要知道所执行的一般性行为既可。由于多态性，需要记住的三件事情减少到一件，这个程序相当平常，但在扩展这个概念后，会发现多态性是怎样帮助你管理非常复杂的程序的。

总之，重载一个函数，只要声明它的不同版本既可。编译器完成剩余的工作。要重载一个函数时必须注意一个重要的限制：两个函数不能只是在返回值上不同，它们必须在参数类型或个数上有所不同（返回类型的不同不能在所有的情况下为编译器确认函数版本提供足够的信息）。

下面是使用函数重载的另一个例子：

```
#include "iostream.h"
#include "stdio.h"
#include "string.h"

void stradd(char *s1, char *s2);
void stradd(char *s1, int si);

main()
{
  char str[80];

  strcpy(str,"您好");
  stradd(str,"张先生");
  cout << "str =" << str << "\n";

  stradd(str,100);
  cout << "str =" << str << "\n";

  return 0;
}

// 连接两个字符串
void stradd(char *s1,char *s2)
```

```
{
  strcat(s1,s2);
}

// 连接一个字符串化的整数字符串
void stradd(char *s1,int i)
{
  char temp[80];

  sprintf(temp,"%d",i);
  strcat(s1,temp);
}
```

在该程序中，函数 stradd()被重载。一个版本完成两个字符串的连接(如图 strcat()函数)，另一个版本字符串化一个整数并将其添加到一个字符串上。这里使用重载创建一个界面，它可以将一个字符串或一个整数添加到另一个串上。

可以使用同一个名字重载不相关的函数，但不应该这样做。例如，可以使用函数名 sqr()创建函数，使其可以返回一个整型数的平方而返回一个双精度数的平方根。然而，这两个操作在本质上是不同的，这种使用方式破坏了函数重载的目的(而且实际上这是一种非常糟糕的编程风格)。在实际中，只应该重载那些非常接近的函数。

1.6 操作符重载

C++还通过操作符重载来实现多态性。众所周知，在C++可以使用操作符<<和>>执行控制台 I/O。它们可以执行这些额外的操作是因为它们在头文件 iostream.h 中被重载。当一个操作符被重载后，它将在特定的类上使用附加的含义，但其他时候它仍然保持其原始的含义。

总得来说，大多数C++的操作符可以在某个特定的类上被重载。例如，回忆前面建立的 stack 类的例子，可以在 stack 类型上重载+操作符，使其可以将一个堆栈中的内容添加到另一个中。然而，对于其他的数据类型，+操作符仍然保持其原始的含义。

因为操作符的重载实际上比函数的重载更复杂，其详细的例子请参考第四天讲述的内容。

1.7 继承

今天前面已提到过，继承是面向对象程序设计的一个重要的特征。在C++中，是通过允许一个类可能将另一个类加入到其声明中来实现的。继承可以创建出一个拥用层次结构的类，从最一般到最具体。该过程首先创建一个基类，它包含该基类所派生出的对象的共同的性质，基类表示了最一般的描述。从基类中派生出的类称为派生类，一个派生类包含所有基类的特征并加入了该派生类特有的性质。下面的例子中创建了一组描述不同类型的建筑物的类。

首先，声明了 building 类，它将作为另外两个派生类的基类。

```
class building {
  int rooms;
  int floors;
  int area;
public:
  void set_rooms(int num);
  int get_rooms();
  void set_floors(int num);
  int get_floors();
  void set_area(int num);
  int get_area();
};
```

因为(对于本例)所有的建筑物有三个一般的特征：一个或多个房间，一层或多层和一个总面积。building 类在声明中体现了这几个方面。以 set 打头的成员函数设置私有成员的值，以 get 打头的成员函数返回这些值。

现在可以使用这个建筑物的定义模式创建描述特定类型建筑物的派生类。例如，下面是一个叫做 house 的派生类：

```
// house 从 building 派生
class house : public building {
  int bedrooms;
  int baths;
public:
  void set_bedrooms(int num);
  int get_bedrooms();
  void set_baths(int num);
  int get_baths();
};
```

注意 building 类是怎样被继承的。继承的一般形式是

```
class 新的类名:访问符被继承类 {
    //新类的定义体
```

}这里，访问指明符是可选择的。然而，如果出现，它必须是关键字 public，private 或 protected(这些选项将在第二天"类与对象"中做进一步学习)。现在，所有的被继承类都使用 public 选项。使用 public 意味着所有基类中的公用成员在继承类中也是公用的，因此在本例中，house 类中的成员可以访问 building 类中的成员函数就像它们是在 house 中声明的一样。然而，house 类的成员函数不能访问 building 类中的私有部分，这是很重要的一点。即使 house 类继承了 building 类，它也只能访问 building 类中的公用部分。这样，继承将不会破坏面向对象程序设计必须的封装原则。

切记：派生类可以直接访问它的成员函数和基类中公用的成员函数。

下面的程序描述了类继承。它使用继承创建了 building 类的的两个派生类：一个是 house 类，一个是 school 类。

```
#include "iostream.h"

class building {
  int rooms;
  int floors;
  int area;
public:
  void set_rooms(int num);
  int get_rooms();
  void set_floors(int num);
  int get_floors();
  void set_area(int num);
  int get_area();
};

// house 从 building 派生
class house : public building {
  int bedrooms;
  int baths;
public:
  void set_bedrooms(int num);
  int get_bedrooms();
  void set_baths(int num);
  int get_baths();
};

// school 也从 building 派生
class school : public building {
  int classrooms
  int offices;
public:
  void set_classrooms(int num);
  int get_classrooms();
  void set_offices(int num);
  int get_offices();
};

void building::set_rooms(int num)
{
  rooms = num;
}

void building::set_floors(int num)
```

```
{
  floors = num;
}

void building::set_area(int num)
{
  area = num;
}

int building::get_rooms()
{
  return rooms;
}

int building::get_floors()
{
  return floors;
}

int building::get_area()
{
  return area;
}

void house::set_bedrooms(int num)
{
  bedrooms = num;
}

void house::set_baths(int num)
{
  baths = num;
}

int house::get_bedrooms()
{
  return bedrooms;
}

int house::get_baths()
{
  return baths;
}
```

```
void school::set_classrooms(int num)
{
  classrooms = num;
}

void school::set_offices(int num)
{
  offices = num;
}

int school::get_classrooms()
{
  return classrooms;
}

int school::get_offices()
{
  return offices;
}

main()
{
  house h;
  school s;

  h.set_rooms(12);
  h.set_floors(3);
  h.set_area(4500);
  h.set_bedrooms(5);
  h.set_baths(3);

  cout << "房子有" << h.get_bedrooms();
  cout << "个卧室\n";

  s.set_rooms(200);
  s.set_classrooms(180);
  s.set_offices(5);
  s.set_area(25000);

  cout << "学校有" << s.get_classrooms();
  cout << "间教室\n";
  cout << "它的面积是" << s.get  area();
```

```
    return 0;
}
```

如程序所示，继承的最大的优点是可以创建一个最一般的类型，而它可以被加入到一个更具体的类型中。这样，每一个对象都可以精确地描述它所属的类型。

在C++的一些文献中，单词基类和派生类用来描述继承。然而，有些文献中可能会使用父类和子类来描述继承。

除了可以提供一个有层次类外，继承还可能通过虚函数的机制提供对运行时多态性的支持(参见第六天的主题"虚函数与多态性")。

1.8 构造函数与析构函数

在很多时候，一个对象的某些部分在使用之前要进行初始化。例如，今天创建的stack类，在该类的对象使用前，成员tos需要被初始化为零，这些是通过函数init()完成的。因为对于初始化的需求是如此普遍，C++允许对象在创建时对自己进行初始化。这种自动的初始化是通过使用构造函数来完成的。

一个构造函数是类中一个特殊的成员函数，它的名字同该类的名字相同。例如，下面的例子中stack类使用了一个构造函数进行初始化：

```
// 下面创建类 stack
class stack {
  int stck[SIZE];
  int tos;
public:
  stack();  // 构造函数
  void push(int i);
  int pop();
};
```

注意，构造函数stack()没有指明返回的类型。在C++中，构造函数不能指定回值。

stack()函数的代码如下：

```
//堆栈的构造函数
stack::stack()
{
tos=0;
cout<<"堆栈初始化\n";
}
```

注意：输出的"堆栈初始化"信息只是为描述构造函数。在实际应用中，大多数的构造函数不会有任何输出或输入，它们只是执行变量的初始化。

一个对象的构造函数在对象创建时被调用。这就是说，它在对象表明被选择时调用。这是C的声明同C++的声明的一个重要的差别。在C中，不严格地说，变量的声明是被动的，大部分在编译时解决。换句话说，在C中变量的声明并非可执行语句。然而，在C++中，变量的声明是一个主动的语句，实际上在运行时执行。一个原因是一个对象的声明需要调用构造函数，因此使其成为可执行语句。尽管这个差别看上去很难把握而且不很实际，但它同变