21 世纪高等院校计算机教材系列

C++ 语言和面向对象程序设计教程习题解答及上机实践

宛延闿　甄　炜　李　俊　等编著

机 械 工 业 出 版 社

本书是与《C++语言和面向对象程序设计教程》（宛延闿等编著）一书配套使用的所有14章习题的详尽解答及上机练习。习题内容丰富、编排精炼准确。习题类型有填空题、选择题、改错题、问答题和编程题等。通过解题和上机可加强读者对C++语言基本概念、面向对象程序设计和面向对象建模内涵的理解。

本书可作为高等院校学生学习C++语言和面向对象程序设计的补充教材，也是广大读者和科技工作者学习C++语言和面向对象程序设计必备的参考书。

图书在版编目（CIP）数据

C++语言和面向对象程序设计教程习题解答及上机实践/宛延闿等编著. —北京：机械工业出版社，2005.6
（21世纪高等院校计算机教材系列）
ISBN 7-111-16159-9

Ⅰ.C++… Ⅱ.宛… Ⅲ.C++语言—程序设计—高等学校—教学参考资料 Ⅳ.TP312

中国版本图书馆CIP数据核字（2005）第012151号

机械工业出版社（北京市百万庄大街22号 邮政编码100037）
策 划：胡毓坚 责任编辑：孙 业 版式设计：张世琴
责任印制：
印刷厂印刷·新华书店北京发行所发行
2005年7月第1版第1次印刷
787mm×1092mm 1/16·8.25印张·198千字
0001—册
定价：元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
本社购书热线电话（010）68326294
封面无防伪标均为盗版

出 版 说 明

计算机技术是一门迅速发展的现代科学技术，它在经济建设与社会发展中，发挥着非常重要的作用。近年来，我国高等院校十分注重人才的培养，大力提倡素质教育、优化知识结构，提倡大学生必须掌握计算机应用技术。为了满足教育的需求，机械工业出版社组织了这套“21 世纪高等院校计算机教材系列”。

在本套系列教材的组织编写过程中，我社聘请了各高等院校相关课程的主讲老师进行了充分的调研和细致的研讨，并针对非计算机专业的课程特点，根据自身的教学经验，总结出知识点、重点和难点，一并纳入到教材中。

本套系列教材定位准确，注重理论教学和实践教学相结合，逻辑性强，层次分明，叙述准确而精炼，图文并茂，习题丰富，非常适合各类高等院校、高等职业学校及相关院校的教学，也可作为各类培训班和自学用书。

参加编写本系列教材的院校包括：清华大学、西安交通大学、北京交通大学、北京邮电大学、北京化工大学、北京科技大学、山东大学、首都经贸大学等。

机械工业出版社

前　言

本书是与《C++语言和面向对象程序设计教程》一书配套使用的习题解答及上机练习。

为了使读者更好地掌握并学会C++语言，每章的习题对理解面向对象程序设计基本概念、C++编程和面向对象建模都是至关重要的。本书的习题类型有填空题、选择题、改错题、问答题以及编程题等，内容丰富、习题设计精炼准确。这些习题侧重点在于对C++语言基本概念、面向对象程序设计和面向对象建模精髓的理解，以便广大读者和学生尽快地全面掌握。鉴于读者或许不一定能够正确完整地回答书中的每道练习题，所以，我们把《C++语言和面向对象程序设计教程》一书中14章的所有习题尽量详尽地解答出来，供广大读者和学生学习时参考。

上机实践部分包括了函数应用、类与对象、数组、指针和字符串、多态性、流类库的输入/输出等，一共26道上机练习题。读者上机的环境既可以采用VC++，也可以采用Borland C++。具体的上机实践环境读者可参照相应的参考书。

参加本书编著的还有石良秀、乔立琴、李保林、代宁、郭应中、高晓丽、王子滨、王浩枫、潘京、季英洁、王□和王心颖等。

宛延闿

目　　录

第一部分　习 题 解 答

第 1 章　绪　　论

1.1　填空题

⑴ 计算机软件开发一直受到_____和_____难题的困扰。

⑵ 面向过程语言主要有_____和_____缺陷。

⑶ 无保留库的软件开发方法把_____活动的高墙拆开。它的开发过程是采用_____方法以及_____和_____。

⑷ 面向对象技术的要点是_____、_____、_____、_____、_____和_____。

⑸ 软件的鲁棒性表明了_____的能力。

⑹ 面向对象技术 OMT 包括了刻画一个系统的三种模型是_____、_____和_____。

答：⑴ 如何超越程序复杂性障碍，如何在计算机系统中自然地表示客观世界

⑵ 数据抽象能力差，模块化能力差

⑶ 三个，面向对象分析和设计方法，面向对象程序设计语言，面向对象数据库

⑷ 抽象，封装，数据和行为联合，共享，对象结构，协同作用

⑸ 系统抵御冲击、维持生存的能力

⑹ 对象模型，功能模型，动态模型

1.2　选择题 (至少有一个，可以多选)

⑴ 零件、配件与设计图的关系，相当于对象与_____的关系。

a. 成员函数；b. 类；c. 运算符；d. 数据项；e. 基本操作；f. 基本属性。

⑵ 传统的软件开发包含了_____。

a. 两个独立的活动，它们是分析和设计以及程序设计；

b. 三个独立的活动，它们是分析和设计，建立模型以及数据库设计；

c. 三个独立的活动，它们是分析和设计，程序设计以及建立概念模型和存取数据库；

d. 四个独立的活动，它们是分析和设计，程序设计，定义概念模型以及存取数据库。

⑶ 对象是_____。

a. 算法 + 数据结构的累加；

b. 算法 + 数据结构 + 程序设计的累加；

c. 把数据结构 + 相应的行为封装在一起的实体；

d. 把数据结构 + 行为 + 函数运算符封装在一起的实体。

(4) 以下哪种是我们要使用面向对象语言的正确理由？_____。

a. 可以自行定义所需的数据类型；

b. 面向对象语言的语句比面向过程语言的语句简单易学好用；

c. 面向对象语言能够自动地发现并修改已发生的错误；

d. 面向对象语言很容易概念化、可重用。

答：(1) b (2) c (3) c (4) a，d

1.3 什么是对象 (object)？什么是面向对象 (object-oriented)？为什么要面向对象？

答：什么是对象？

在现实生活中，文档中的一个段落，工作站上的一个窗口，国际象棋中的皇后，一粒米，二叉树，小乔的自行车，小石的汽车，小李的房子，策略符号表等等都是对象。

一个对象像一个软件构造块，它包含了数据结构和提供的相关的行为 (操作)。在计算机中表示真实世界模拟思维的抽象。对象本身可以为用户提供一系列服务：改变对象的状态、测试、传递消息等，用户无需知道服务的任何实现细节，操作完全是封闭的。

什么是面向对象？

“面向对象”是把一组相互无关联的对象有机地集成在一起的软件，而这些对象都是将数据结构和行为紧密结合在一起，这与传统的将数据结构与行为松散连接的模式完全不一样。

为什么要面向对象？

四十年以来的研究与开发，突出的发现：软件开发仅仅把注意力放在功能上是远远不够的。诚然，软件开发首要的任务是提供正确的功能，但在功能结构上，必须能够扩充、删除和修改，软件必须能够反复使用 (可重用)。功能本身是易变的，不可能达到上述目的。而对象则在客户需求不断变化的情况下相对稳定，因为无论功能怎么千变万化，一个问题空间中的对象一般总能保持其稳定不变性。这样，围绕对象构造的软件系统自然也会有较好的稳定性。所以，我们的软件系统要采用面向对象技术。

1.4 什么是面向对象程序设计 (OOP)？什么是面向对象程序设计语言 (OOPL)？

答：什么是面向对象程序设计 (OOP)？

OOP 中心是围绕抽象数据类型和类、类层次结构、继承和多态进行的程序设计工作。类和继承是符合人们一般思维方式的描述模式。

什么是面向对象程序设计语言 (OOPL)？

我们说，具备以下特征的语言，就是 OOPL。这些特征有：①对象；②可编程性；③访问控制；④继承性；⑤多态性；⑥可预见性。

1.5 自顶向下设计方法与自底向上设计方法各自有什么特点？这两种设计方法与面向对象方法有何关联？

答：自顶向下设计方法的特点：从问题大的方面入手来寻找解决办法，然后再解决剩下相对较小的问题。这样，经过多次迭代最终获得完整明确的解决办法。

自底向上的设计方法的特点：从解决基本的、简单的问题入手。在此基础上逐步建立解决复杂问题的能力，直至整个问题得以圆满解决。正好与自顶向下的设计方法相反。

面向对象设计方法是兼有这两者（自顶向下和自底向上）特点的方法。它鼓励人们从问题的基本的简单的方面入手，用对象来描述问题的主要方面（这正是自底向上方法的本质）。同时，面向对象设计方法又要求人们面向目标，考虑为了达到这一目标该如何建立这些基本对象（这也正体现了自顶向下的设计思想）。面向对象设计方法关键的优点在于代码的共享和重用，这是问题的基本方面，正是因为此特点，就有别于自顶向下和自底向上设计方法。

1.6 面向对象方法学包括几个基本重要阶段？如何刻画一个软件系统？

答：面向对象方法学是由应用领域中所创建的模型组成的，然后在系统设计期间添加实现的细节。此方法称作对象模型技术（OMT）方法。有以下几个重要阶段：

⑴ 分析阶段；

⑵ 系统设计阶段；

⑶ 对象设计阶段；

⑷ 实现阶段。

OMT 方法采用三种模型来刻画一个系统：对象模型，它描述系统中的对象和它们之间的关系；动态模型，它描述系统中对象之间的相互作用；功能模型，它描述系统中的数据的变迁。每种模型应用于开发的不同阶段，以几种不同的观点来刻画不同的模型。一个系统的完全描述需要包含这三种模型。

1.7 为什么要学习 C++ 语言？它在企业信息化建设中起什么样的作用？

答：为什么要学习 C++ 语言？

首先，C++ 是一种混合型的面向对象语言，它既保留了高效的 C，又具备了强大的面向对象特征。其次，C++ 是世界流行语言，是世界各大软件公司开发软件的主流语言，功能强大，并能把各大软件公司经过几十年用 C 开发的类库和有用的软件保留下来（减少花费），同时又能在此基础上开发先进的面向对象软件。所以，C++ 是软件开发的很好的起跳点。

C++ 在企业信息化建设中起什么样的作用？

作用很大。目前，C++ 在尖端信息领域，如金融、航天航空、通信等领域已站稳脚跟，并逐步在网络、移动通信、大型数据处理、石化、电子等企业，为开发人员所接受。我国信息化建设在计算机语言方面，C++ 已处主导作用。

1.8 软件从广义角度可分为哪几种？试举例说明。

答：软件从广义角度可分为系统软件和应用软件。

系统软件包括操作系统，比如 Windows，UNIX 等。

应用软件，种类繁多。比如字处理软件 WordPerfect，Word，AmiPro；电子表格应用软件有 Execl；画图的软件有 AutoCAD，Visio 等。

第 2 章　C++ 程序设计初步

2.1　填空题

⑴ C++ 程序通常使用_____程序进入计算机。

⑵ 在 C++ 系统中，编译开始之前执行的是_____程序。

⑶ _____程序把编译器的输出与不同的库函数组装在一起产生可执行代码。

⑷ _____程序把内存中各部分 C++ 可执行代码组装在一起。

⑸ 每个 C++ 程序开始执行的函数是_____。

⑹ 每个函数体由_____符号开始，又以_____符号结束函数体。每个语句结束用_____符号。

⑺ 将数据和函数捆绑在一起称为_____。

⑻ 对象的两个主要成份是_____和_____函数。

答：⑴ 编辑 (editor)　⑵ 预处理　⑶ 连接 (linker)

⑷ 装入 (loader)　⑸ main ()　⑹ {,} , ;

⑺ 封装　⑻ 数据，对数据的操作

2.2　选择题 (至少选一个，可以多选)

⑴ 对象和类的关系，好像是_____。

a. 变量和数据类型的关系；

b. 变量和成员函数的关系；

c. 成员函数和基本操作的关系；

d. 成员函数和非成员函数的关系。

⑵ 如果一种语言能够产生新的数据类型，那么我们就称为_____。

a. 是错误的；　b. 是可封装的；

c. 是可继承的；　d. 是可扩展的。

⑶ endl 是流操作符，它是输入/输出流库的一部分，其含义为_____。

a. end linker (结束连接)；　b. end loader (结束装入)；

c. end line (输出一个新行)；　d. end level (结束层次)。

⑷ 注释是程序员理解阅读程序的良好工具。C++ 注释行有以下几种：_____。

a. 单行注释行//一种；

b. 单行注释行//和多行注释行/*…*/两种；

c. 没有相应的注释行；

d. 有三种注释行：// (单行注释)，/*…*/ (多行注释)，/*…**/ (多行注释)。

答：⑴ a　⑵ d　⑶ c　⑷ b

2.3　C++ 对 C 作了哪些重要扩充？

答：主要有以下 16 条重要扩充，它们是：

① 分程序内的说明：允许在分程序内和在可执行代码之后出现变量声明。

② const 说明符：既可用于一个实体在其作用域内的值说明为常量，也可用于对指针指向的数据。函数的参数也可以说明为 const。

③ sizeof 运算符，扩展了使用范围。以下都是合法的：sizeof (数据类型)，sizeof (变量)，sizeof 数据类型，sizeof 变量。

④ 强制类型转换，支持两种格式：

类型 (表达式)；

(类型) 表达式；//C 语言支持的

⑤ 内联 (line) 函数：把指定的函数直接插入到每个调用语句处，以增加内存为代价，加快程序执行速度。

⑥ 缺省参数：函数说明时，可以把参数中的后面 1 个或几个赋以缺省值。调用时也可以不提供这些缺省的自变量。

⑦ 引用参数：在参数类型后加一“&”，表示引用参数——即成为自变量的别名。

⑧ 函数名重载：体现了多态性。

⑨ new 和 delete 运算符进行内存动态分配和释放。

⑩ void 指针和 void 函数，表示不返回任何值。

⑪ 对象 (object)：略。

⑫ 类 (class)：略。

⑬ 成员函数，它是类在数据上的操作。

⑭ 构造函数和析构函数：前者完成对象的初始化工作，后者释放该对象所占的内存资源。

⑮ 继承：有单一继承和多重继承。

⑯ 命名空间和例外处理：是 C++ 国际标准中重要组成部分，扩大了应用范围和出错处理机制。

2.4 试编写一个 C++ 程序。要求从键盘输入三个整数，并打印这些数的和 (sum) 以及这三个整数的平均值 (average)。屏幕对话应出现以下格式的画面：

Input three different integers：13 27 14

Sum is 54

Average is 18

答：本题的 C++ 源程序如下：

```
#include <iostream>
using namespace std;
int main() {
  int x,y,z,sum,average;
  cout << "Input three different integers:";
  cin >> x >> y >> z;
  cout << "Sum is" << x + y + z << endl;
  cout << "Average is" << (x + y + z)/3 << endl;
  return 0;
}
```

当输入 13 27 14 **之后，就在屏幕上出现以下画面：**

Input three different integers：13 27 14

Sum is 54

Average is 18

2.5 **试编写一个产生以下输出的** C++ **程序：**

20

40

39

要求：使用一个值为 20 **的整型常量并施加算术赋值运算符产生** 40**，然后使用递减运算符产生** 39**。**

答：本题的 C++ **源程序如下：**

```
//算术赋值和递减
#include <iostream>
using namespace std;
int main() {
  int Variable=20;
  cout<<Variable<<endl;     //Variable 是 20
  Variable *=2;             //Variable 变成为 40
  cout<<Variable--<<endl;   //先显示 40,然后再减掉 1
  cout<<Variable<<endl;     //Variable 现在是 39
  return 0;
}
```

2.6 **试用** C++ **程序，产生如下格式表格：**

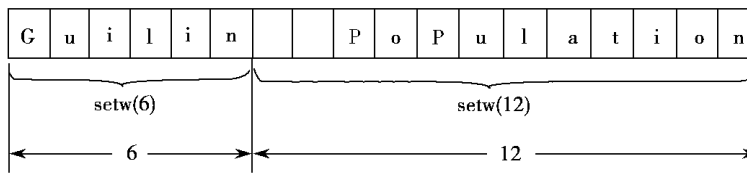
```
2005      135
2006     8280
2007    11330
2008    32320
      |←——8——→|
```

要求：按上述格式表格输入、对齐；所有的输出只能用一条 cout **语句实现。**

提示：为了完成上述要求，就必须用到 setw **操作符。** setw **操作符可使得流中的数字（或字符串）以** n **个字符的宽度输出。** setw (n) **中的** n **为参数，值在字段中是右对齐的。用** setw (n) **可以改变输出的字段的宽度。例如，下面语句**

```
cout<<setw(6)<<"Guilin"<<setw(12)<<"Population"<<endl;
```

其输出为：

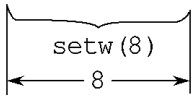


答：本题 C++ 源程序如下：

```
//生成规定格式的表格
#include <iostream>
#include <iomanip>
using namespace std;
int main() {
  cout <<2005 <<setw(8) <<135 <<endl
       <<2006 <<setw(8) <<8280 <<endl
       <<2007 <<setw(8) <<11330 <<endl
       <<2008 <<setw(8) <<32320 <<endl;
  return 0;
}
```

输出结果为

```
2005     135
2006    8280
2007   11330
2008   32320
```



第 3 章　C++ 语言基础

3.1　**填空题**

⑴ enum 类型中前三个枚举元的值通常是_____、_____和_____。

⑵ C++ 编译器会忽略掉空白符这一规则的两个例外情况是_____和_____。

⑶ 告诉计算机该做什么的 C++ 命令，称之为_____。

⑷ 库函数的实际代码包含在_____文件中。

⑸ 数组中的所有元素必须是_____数据类型。

⑹ C++ 字符串是一个_____类型的_____。

⑺ 在源文件中使用的数组名称表示数组的_____。

⑻ 所有程序都是根据三种类型的控制语句编制的，它们是_____、_____和_____。

⑼ 当条件为真时执行一个动作，否则执行另一个动作，是用_____选择结构。

⑽ 指定具体次数的循环称之为_____循环。

答： ⑴ 0，1，2　　⑵ 字符常量，预处理指令

⑶ 语句　　⑷ 库　　⑸ 相同的

⑹ 数组，字符　　⑺ 内存地址

⑻ 顺序，选择，循环　　⑼ if/else

⑽ 计算控制或有限

3.2　**选择题（至少选一个，可以多选）**

⑴ 关系运算符_____。

a. 是逻辑运算两个操作数，仅此而已；

b. 是将一个操作的结果赋予另一个操作数；

c. 比较两个操作数的大小，要么相等，要么不等；

d. 比较的值的类型是任何 C++ 内置类型，并产生一个布尔结果。

⑵ 在一个包含多个语句的循环体的 for 循环中，分号应该出现在_____后面。

a. for 语句自身；　　b. 循环体中的每个语句；

c. 循环体的终边花括号；　　d. 测试表达式。

⑶ 访问数组元素要采用_____。

a. 先进后出的方法；　　b. 先进先出的方法；

c. 点运算符；　　d. 数组下标；　　e. 成员名称。

⑷ 访问多维数组时，每个数组下标_____。

a. 用分号分隔开；

b. 用方括号围住再用分号分隔开；

c. 用分号分隔开再用方括号围住；

d. **用方括号围住；**

e. **用逗号分隔开。**

⑸ **试问** intArray [8] **是指的数组中哪一个元素？**_____**。**

a. **第** 8 **个元素；**

b. **第** 7 **个元素；**

c. **第** 9 **个元素；**

d. **不确定。**

⑹ string **类的对象**_____**。**

a. **以空字符作为终止标志；**

b. **无需进行内存分配管理；**

c. **可使用赋值运算符进行拷贝；**

d. **它没有成员函数。**

⑺ exit () **库函数退出**_____**。**

a. **它所在的函数；**

b. **它所在的循环；**

c. **它所在的块；**

d. **它所在的程序。**

⑻ **逻辑运算符** && **和** || **：**_____**。**

a. **比较两个数字值的关系；**

b. **运算两个数字值；**

c. **比较两个布尔值的关系；**

d. **运算两个布尔值。**

⑼ break **语句退出**_____**。**

a. **仅仅是最内层的循环；**

b. **仅仅是最内层的** switch **语句；**

c. **所有循环和** switch **语句；**

d. **最内层的循环或** switch **语句。**

⑽ **块内定义的变量其可见范围是**_____**。**

a. **从定义点向前看的函数部分；**

b. **从定义点向前看的程序部分；**

c. **从定义点向前看的块的部分；**

d. **整个函数区域。**

答： ⑴ c，d ⑵ b，d ⑶ d ⑷ d ⑸ c

⑹ b，c ⑺ d ⑻ d ⑼ d ⑽ c

3. 3 **试写出** 4 **种不同的** C++ **语句，使其对整型变量** x **都加** 1**。**

答：这 4 **种不同的** C++ **语句是：**

x = x + 1；

x += 1；

++ x；

```
    x++;
```

3.4 判断下面程序片段是否正确。如果错误，则要说明错在哪里，并用正确方式加以更正。

(1)
```
while(c<=5){
    product *= C;
      ++C;
```

(2)
```
cin >> value;
```

(3)
```
if(gender==1)
    cout << "People" << endl;
else;
  cout << "Student" << endl;
```

(4)
```
x=1;
while(x<=10);
    x++;
}
```

(5)
```
for(y=0.1;y!=1.0;y+=0.1)
    cout << y << endl;
```

(6)
```
switch(n){
    case 1:
      cout << "The number is 1" << endl;
    case 2:
      cout << "The number is 2" << endl;
      break;
    default:
      cout << "The number is not 1 or 2" << endl;
      break;
}
```

(7) 下面的代码应当打印 1 到 10。
```
    n=1;
    while(n<10)
      cout << n++ << endl;
```

答：(1) 错误：丢了 while 循环的右花括号“}”。
更正：在 ++C；之后添上右花括号} 。

(2) 错误：流输入符号 << 用错了。
更正：把 << 改为 >> 。

(3) 错误：在 else 后面用了分号“;”产生了逻辑错误。这样第二个输出语句永远在执行着。
更正：去掉 else 后面的分号“;”。

(4) 错误：在 while 头语句后面的分号“;”将产生无限循环。
更正：去掉 while 语句后面的分号“;”，加上 {。或者去掉分号“;”，并去掉

“x ++ ;”后面的右花括号} 。

⑸ 错误：使用了浮点数去控制一个 for 循环结构。

更正：应使用整型数来控制执行 for 循环。

正确的语句为：

```
for(y =1;y!=10;y ++)
    cout << ((float)y/10) << endl;
```

⑹ 错误：在 case 1：处丢了一个 break 语句。

更正：添加上 break 语句。请读者注意：这不是必须的。如果程序员要求每次执行完 case 1：语句后都要执行 case 2：语句的话，那就不是一个错误。

⑺ 错误：在 while 循环中的关系运算符不合适。

更正：应该把 < 改为 <= ，并把 10 改为 11。

3. 5 试用穷举法找出 1 至 100 之间的质数并把它们显示出来。要求分别使用 while，do - while 和 for 循环语句加以实现。

答：1. 用 while 循环语句实现

```
#include < iostream >
using namespace std;
void main() {
    int i,j,k,signal;
    i =2;
    while(i <=100) {
      signal =1;
      k = i/2;
      while(j <= k) {
        if(i% j ==0) {
            signal =0
            break;
        }
        j ++;
      }
    if(signal)
      cout << i << "    ";
    i ++;
  }
}
```

2. 用 do [CD * 3] while 循环语句实现

```
#include < iostream >
#include < math >
using namespace std;
void main() {
```

```
    int i,j,k,signal;
    i =2;
    do {
        signal =1;
        k = i/2;
        j =2;
        do {
            if(i% j ==0) {
              signal =0
              break;
            }
          j ++;
        } while(j <= k);
        if(signal)
          cout << i <<"    ";
        i ++;
    } while(i <=100);
}
```

3. **用** for **循环语句实现**

```
#include < iostream >
using namespace std;
void main() {
  int i,j,k,signal;
  for(i =2;i <=100;i ++) {
      signal =1;
      k = i/2;
      for(j =2;j <= k;j ++) {
          if(i% j ==0) {
              signal =0;
              break;
          }
      }
      if(signal)
      cout << i <<"    ";
    }
}
```

4. **程序运行结果显示为：**

2 3 5 7 11 13 17 19 23 29 31 37 41 43
47 53 59 61 71 73 79 83 97

3.6 试编写一个四则运算计算器的 C++ 模拟程序。要求用户输入一个数、一个运算符和另一个数(使用浮点数),然后程序执行指定的算术运算:加(+)、减(-)、乘(*)和除(/)这两个数。只允许使用一个 switch 语句来选择操作,最后显示结果。

当本次计算完成时,程序应询问用户是否继续进行下一次计算,回答是'y'或'n'。

其样本输出为:

请输入第一个数,运算符,再输入第二个数:10/3

回答是 3.333333

是否继续下一个计算(y/n)?y

请输入第一个数,运算符,再输入第二个数:32+100

回答是 132

是否继续下一个计算(y/n)?n

答://模拟四则运算的计算器

```
#include <iostream>
using namespace std;
int main(){
    double num1,num2,answer;
    char oper,ch;
    do{
      cout<<"\n请输入第一个数,运算符,再输入第二个数:";
      cin>>num1>>oper>>num2;
      switch(oper){
        case'+':answer=num1+num2;break;
        case'-':answer=num1-num2;break;
        case'*':answer=num1*num2;break;
        case'/':answer=num1/num2;break;
        default:answer=0;
    }
    cout<<"回答是"<<answer;
    cout<<"\n是否继续下一个计算(y/n)?";
    cin>>ch;
  }while(ch!='n');
  return 0;
}
```

程序运行结果为:

请输入第一个数,运算符,再输入第二个数:10/3

回答是:3.333333

是否继续下一个计算(y/n)?y

请输入第一个数,运算符,再输入第二个数:32+100

回答是 132

是否继续下一个计算（y/n）？n

提醒读者的是：输入的数可以是浮点数。在上述样本输入中并没有使用浮点数。读者可以用浮点数重新试一下。因为在 C++ 程序中，我们已把 num1，num2 和 answer 均说明为 double 类型了。

第 4 章　C++ 函 数

4.1　**填空题**

⑴ C++ 中的程序模块被称为_____和_____。

⑵ 一个函数是用_____方式被引用。

⑶ 只在函数内定义的变量被称为_____。

⑷ 在调用函数时_____语句用于传递返回调用函数的表达式值。

⑸ 函数定义的第一行被称为_____。

⑹ 不返回任何东西的函数的返回值为_____。

⑺ 只有一条语句的函数描述被称做函数_____或者函数_____。

⑻ 实现函数功能的语句组成了函数_____。

⑼ 在 C++ 中，允许不同的函数具有相同的名字，只要它们有不同的类型和不同的参数个数。这种函数被称为_____。

⑽ 函数_____能够使得单一函数定义并执行很多不同数据类型的任务。

答：⑴ 函数，类　⑵ 函数调用　⑶ 局部变量

⑷ return　⑸ 声明　⑹ void

⑺ 声明，原型　⑻ 体　⑼ 重载函数

⑽ 模板

4.2　**找出下面语句片段中的错误，并加以改正。**

⑴
```
int g(void) {
    cout << "Inside function g" << endl;
    int h(void) {
        cout << "Inside function h" << endl;
    }
}
```

⑵
```
int sum(int n) {
    if(n == 0)
        return 0;
    else
        n + sum(n - 1);
}
```

⑶
```
int sum(int x, int y) {
    int result;
    result = x + y;
}
```

⑷
```
void f(float a); {
```

```
    float a;
    cout << a << endl;
}
```

⑸
```
void product(void) {
  int a, b, c, result;
  cout << "Enter three integers:";
  cin >> a >> b >> c;
  result = a * b * c;
  cout << "Result is" << result << endl;
  return result;
}
```

答： ⑴ **错误：函数** h() **被定义在函数** g() **中。**

更正：把函数 h() **移到函数** g() **的定义外面去。**

⑵ **错误：不会返回** n + sum(n - 1) **的结果，**sum **返回不合适不正确的结果。**

更正：把 else **语句下一行重写为：**

return n + sum(n - 1)；

⑶ **错误：该函数支持返回一个整数(**int**)值，但它不是整数。**

更正：将 result = x + y **一行删除，并改为**

return x + y；

⑷ **错误：在右边圆括号后面的分号“；”是不对的。**float a**；导致重新定义参数** a**。**

更正：删除函数右边圆括号后面的分号，并删除 float a**；**

⑸ **错误：因为在函数定义前头用了** void**，所以该函数是不返回任何值的。即不支持返回值，用** return **语句是错误的。**

更正：删去 return **语句即可。**

4.3 **选择题(至少选一个，可以多选)**

⑴ **函数最重要的一个作用是**_____**。**

a. **提供一段代码和代码名称；**

b. **大大减少了程序编制的工作量；**

c. **接收参数并提供返回值；**

d. **有助于以概念单元的形式组织程序。**

⑵ **函数参数是**_____**。**

a. **函数请求调用的程序接收值的变量；**

b. **函数用来阻止接收调用程序的值的方式；**

c. **调用程序发送给函数的值；**

d. **函数返回给调用程序的一种值。**

⑶ **静态局部变量的作用是**_____**。**

a. **使得变量只对一个函数可见；**

b. **使得变量只对一些函数可见；**

c. **当函数没有执行时把变量永远保存在内存中；**

d. 当函数没有执行时保留其值。

⑷ 当通过引用传递参数时，_____。

a. 函数创建一个变量以存储参数的值；

b. 函数无法访问传递引用参数的值，而是自动引用的；

c. 调用程序创建一个临时变量以容纳参数的值；

d. 函数访问调用程序中参数的原始值。

⑸ 缺省参数的值可以_____。

a. 由调用程序提供；

b. 由函数提供；

c. 由全局变量提供；

d. 由局部变量提供。

⑹ 重载函数是_____。

a. 一组具有相同名称的函数；

b. 一组参数的个数和类型都完全相同的函数；

c. 使得程序员的工作愉快又轻松；

d. 使得无法预料的工作能够迎刃而解。

⑺ 内联函数是_____。

a. 一组不能包含任何静态变量，也不能递归的内嵌函数；

b. 与普通的重载函数功能相近的函数；

c. 一组不能包含任何循环语句和 GOTO 语句的函数；

d. 可以进行例外处理，但并不一定要将其定义出现在第一次被调用之前。

⑻ 递归函数是_____。

a. 一组不能包含任何循环语句的函数；

b. 一组通过自己调用自己的方法获得结果的函数；

c. 一组必须采用递归算法才能获得结果的函数；

d. 一组既可以采用递归算法也可以采用非递归算法的函数。

答：⑴ a，c　⑵ c　⑶ a，d　⑷ d

⑸ a，b　⑹ a，c　⑺ a，c　⑻ b，d

4.4 试叙述下面程序中每个元素（包括标识符，变量，函数及函数原型）的作用域（函数作用域，文件作用域，块作用域或函数原型作用域）：

```
#include <iostream>
using namespace std;
int cube(int y);
void main() {
    int x;
    for(x=1;x<=10;x++)
        cout<<cube(x)<<endl;
}
int cube(int y) {
```

```
    return y * y * y;
}
```

答：(1) 在 main () 中的变量 x,它在块作用域中。

(2) 在 cube () 中的变量 y,它在块作用域中。

(3) 函数 cube,它在文件作用域中。

(4) 函数 main () ,它在文件作用域中。

(5) 函数原型 cube () ,它在文件作用域中。

(6) 在函数原型 cube () 中的标识符 y,它在函数原型作用域中。

4.5 给出下面每个函数的函数首行,并给出它们相应的函数原型：

(1) 函数 tennisof 有两个双精度的浮点自变量 sd1,sd2,其结果返回双精度浮点数。

(2) 函数 smallest 有三个整型自变量 x,y,z,其结果返回一个整型数。

(3) 函数 objection 不接收任何自变量,也不返回值。

(4) 函数 floatToInt 有一个整型自变量 number,并返回一个浮点数的结果。

答：(1) double tennisof (double sd1,double sd2)

函数原型为 double tennisof (double,double) ；

(2) int smallest (int x,int y,int z)

函数原型为 int smallest (int,int,int) ；

(3) void objection (void) //在 C++ 中可以写成 (void)

函数原型为 void objection (void) ;//在 C++ 中可以写成 (void)

(4) float floatToInt (int number)

函数原型为 float floatToInt (int) ；

4.6 为什么函数原型能够包含一个参数类型说明为诸如 float& 的方式？

答：因为程序员说明为类型“引用”float 的引用参数,是通过引用调用来访问原始的自变量。

4.7 试用内联函数编写一完整的计算球体积的 C++ 程序。假定该内联函数名称为 sphereVolume,首先提示用户输入球的半径,然后再使用下面赋值语句

```
volume = (4/3) * 3.14159 * pow(radius,3)
```

计算并打印球的体积。

答：

```
//用内联函数计算球的体积
#include <iostream>
using namespace std;
const float PI = 3.14159;
inline float sphereVolume(const float r) {
      return 4.0/3.0 * PI * r * r * r;
int main() {
    float radius;
    cout << "Enter the length of the radius of your sphere:";
    cin >> radius;
    cout << "Volume of sphere with radius" << radius <<
        "is" << sphereVolume(radius) << '\n';
```

```
    return 0;
}
```

4.8 **用函数模板来实现函数 swap（x，y）的交换 x，y 值的功能。要求用整型数和浮点数作出两种交换的结果。**

```
//交换 x，y 的值
#include <iostream>
using namespace std;
template <typename T> void swap(T&d,T&y) {
    T z;
    z = x;
    x = y;
    y = z;
}
void main() {
    int j =1,k =2;
    double v =3.0,w =4.0;
    cout <<"j =" <<j <<"k =" <<k <<endl;
    cout <<"v =" <<v <<"w =" <<w <<endl;
    swap(j,k);      //交换整型数
    swap(v,w);      //交换双精浮点数
    cout <<"交换之后:" <<endl;
    cout <<"j =" <<j <<"k =" <<k <<endl;
    cout <<"v =" <<v <<"w =" <<w <<endl;
}
```

结果输出为：

```
j =1  k =2
v =3.14  w =4.35
```

交换之后：

```
j =2  k =1
v =4.35  w =3.14
```

第 5 章　指针和引用

5.1　填空题

⑴ 指针是一个变量，它包含了另一个变量的_____值。

⑵ 给指针初始化的三个值是_____、_____和_____。

⑶ 赋予指针的整数只能是_____。

⑷ 指向相邻 float 变量的指针的值相差_____字节。

⑸ 地址是_____，而指针是_____。

⑹ 指针的一个重要用途是引用那些没有_____的内存地址。

⑺ * 号放在数据类型后表示_____，* 号放在变量名前表示_____。

⑻ 向函数传递参数有三种方式，其中只有_____传递和_____传递允许函数修改调用者程序。

⑼ 一个 void 类型的指针可以保存指向_____的指针。

⑽ 引用有三种用途，它们是_____、_____和_____。

答：⑴ 地址 (address)　⑵ 0，NULL，一个地址　⑶ 0　⑷ 4 字节　⑸ 常量，变量　⑹ 名称　⑺ 指向，所指变量的值　⑻ 引用，指针　⑼ 任何数据类型　⑽ 作为函数参数，作为返回值，作为独立引用

5.2　试找出下面程序片段中的错误并加以更正。

假定

```
int *zPtr;  //zPtr 是数组 z 的引用
int *aPtr=NULL;
int *zPtr=NULL;
int number,i;
int z[5]={1,2,3,4,5}
sPtr=z;
```

程序片段是：

⑴
```
++zPtr;
```

⑵
```
//用指针获取数组的第一个值
number=zPtr;
```

⑶
```
//把数组元素 2 (第三个值) 赋予 number
number=*zPtr[2];
```

⑷
```
//打印整个数组 z
for(i=0; i<=5; i++)
cout<<zPtr[i]<<endl;
```

⑸
```
//用 sPtr，把指向 sPtr 的值赋予 number
number=*sPtr;
```

⑹
```
char s[10];
cout << strncp(5,"hello", 5) << endl;
```

答：⑴ 错误：zPtr 未被初始化。

更正：用 zPtr = z；来初始化 zPtr。

⑵ 错误：该指针未被间接引用。

更正：更换语句为 number = * zPtr；。

⑶ 错误：zPtr [2] 不是一个指针，所以不应该被间接引用。

更正：把 * zPtr [2] 更改为 zPtr [2] 即可。

⑷ 错误：在该数组之外引用数组元素应受到指针下标的限制。

更正：在 for 循环结构中更换相应的操作符 < （即把 <= 换成 <）以避免数组越界。

⑸ 错误：不能用指针算术运算来修改一个数组名字。

更正：用指针变量替换数组名，以完成指针算术运算，或用数组名的下标引用一个具体元素。

⑹ 错误：对字符数组 S 没有足够大的空间存储终结字符 NULL。

正确：将数组说明改为具有更多的元素的（说明）。

5.3 选择题（至少选一个，可以多选）

⑴ 指针是_____。

a. 一种数据类型；
b. 访问变量的一种标记；
c. 变量的地址；
d. 存储地址的变量；
e. 地址变量的数据类型。

⑵ 表达式 * express 可以说成_____。

a. 引用 express 的内容；
b. 指向 express 的指针；
c. 引用 express 所指变量的内容；
d. 间接引用 express。

⑶ 引用和指针的区别是_____。

a. 引用是一个别名，指针则是存放地址的变量；
b. 引用是一个别名，指针也是一个变量的别名；
c. 引用可以为 NULL 值，指针也可以为 NULL 值；
d. 引用不能为 NULL 值，指针则可赋为 NULL 值；
e. 引用和指针没有什么重大区别。

⑷ 指针所指的变量的类型必须是指针定义的一部分，这是为了_____。

a. 访问结构成员时可以将指针进行加法运算；
b. 对它们执行算术运算时不会把数据类型弄混淆；
c. 没有太说服力的理由；
d. 访问数组成员时编译可以执行正确的算术运算。

⑸ 运算符 * 和 & 的作用是_____。

a. 运算符 * 和运算符 & 的作用基本相似；

b. 运算符 * 是一个指针运算的双目操作符，用来访问指针指向的两个对象的值；

c. 运算符 & 是一个取地址的单目操作符，用来获取一个对象的地址；

d. 运算符 * 是一个指针运算的单目操作符，用来访问指针所指向的对象值；

e. 运算符 & 是一个取地址的双目操作符，用来获取两个对象的地址。

答：(1) a，d (2) a，c，d (3) a，d (4) b，d (5) c，d

5.4 判断下面叙述是否正确 (true)，如果不正确 (false)，则要说明为什么？

(1) 地址操作符 & 只能用于常量、表达式和用存储类 register 说明的变量。

(2) 说明为 void 的指针能被间接引用。

(3) 不同类型的指针如果没有 cast 操作就不能指定另一个。

答：(1) 不正确。地址操作符 & 只能用于变量，而不能用于常量、表达式和用存储类register说明的变量。

(2) 不正确。一个 void 指针不能间接引用，因为无法精确知道应当间接引用多少个字节内存。

(3) 不正确。类型 void 指针能够指定另一个类型的指针，但这种指针是用显式类型 cast 来指定的。

5.5 const int * pointer1 和 int * const pointer2 的区别是什么？

答：const int * pointer1 声明了一个指向整型常量的指针 pointer1，因此不能通过指针 pointer1 来改变它所指向的整型数值。

int * const pointer2 声明了一个指针型常量，用来存放整型变量的地址，这个指针一旦被初始化之后，就不能被重新赋值了。

5.6 字符串"I Love China !"中的结束符是什么？

答：是 NULL 字符。

5.7 试编写一个函数，统计英文一个句子中所包含的字母个数 (提示：用字符指针实现)。

答：//统计英文句子中的字母个数

```
#include <iostream>
using namespace std;
int count(char *str){
    int i,number=0;
    for(i=0;str[i];i++){
       if((str[i]>='a'&&str[i]<='z')
          ||(str[i]>='A'&&str[i]<='Z'))
          number++;
    }
    return number;
}
void main(){
    char text[200];
    cout<<"输入一个英文句子:"<<endl;
```

```
    gets(text);
    cout << "这个英文句子有" << count(text) << "个字母"。
        << endl;
}
```

程序运行输出结果为：

输入一个英文句子：

```
I Love China!
```

这个英文句子有 10 **个字母。**

5.8 **我们希望定义一个雇员类** Employee。**该类包含雇员的姓名、地址、城市和邮编等属性，以及** changeName () 、changeAddress () **和** display () **函数，其中函数** display () **采用** cout **语句来显示姓名、地址、城市和邮编，而函数** changeName () **则可以改变雇员对象的姓名属性，**changeAddress () **可以改变雇员对象的地址属性。**

试编写这个 Employee **类，并用具体雇员信息进行测试。**

答：//**雇员信息管理**

```
#include <iostream>
#include <string>
using namespace std;
class Employee {
  private:
    char name[30];
    char street[30];
    char city[20];
    char zip[6];
  public:
    Employee(char *na,char *str,char *ct,char *z);
    void changeName(char *na);
    void changeAddress(char *str);
    void display();
}
Employee::Employee(char *na,char *str,char *ct,char *z) {
  strcpy(name,na);
  strcpy(street,str);
  strcpy(city,ct);
  strcpy(zip,z);
}
void Employee::changeName(char *na) {
    strcpy(name,na);
}
void Employee::changeAddress(char *str) {
```

```
    strcpy(street,str);
}
void Employee::display() {
    cout << name << "    " << street << "    ";
    cout << city << "    " << zip;
}
void main(void) {
    Employee Emp1("乔立琴","看丹富丰路 6 号","北京","100070");
    Emp1.display();
    cout << endl;
    Emp1.changeName("李保林");
    Emp1.display();
    Emp1.changeAddress("康保大街 9 号");
    Emp1.display();
    cout << endl;
```

程序输出为：

乔立琴　看丹富丰路 6 **号　北京**　100070

李保林　康保大街 9 **号　北京**　100070

第 6 章　类 和 对 象

6.1　填空题

(1) _____保留字引入了结构的定义。

(2) _____操作符能够存取该类的对象；_____操作符能够指向该类的对象。

(3) 初始化一个类的数据成员的专用成员函数是_____。

(4) 对一个类的成员的默认存取是_____。

(5) 通常一个类的成员函数采用_____存取，而一个类的数据成员通常采用_____存取。

(6) _____保留字和_____保留字可以引入一个类的定义。

(7) _____语法被用来初始化一个类的常量成员。

(8) 一个非成员函数必须声明为一个类的_____才有可能存取该类的私有 (private) 数据成员。

(9) _____操作符动态地分配指定类型的一个对象的内存，并返回一个指向那个类型的_____。

(10) 一个对象的成员函数拥有一个指向该对象的指针，我们称它为_____指针。

(11) _____保留字表示在对象或变量初始化后是不可以被修改的。

(12) 如果对一个类的成员对象不提供成员初始化，那么该对象被称为_____。

(13) _____操作符回收曾用 new 操作符动态分配的内存。

(14) 友元函数可以用_____和_____存取一个对象的成员。

(15) 如果成员函数被说明为 static，那么它就不能访问_____的类成员。

(16) 在面向对象程序设计中，当某个对象是另一个对象的属性时，就可能产生_____。

答：(1) struct　(2) 点 (.)，箭头 (->)　(3) 构造函数　(4) private　(5) public，private　(6) class，struct　(7) 成员初始化　(8) friend　(9) new，指针　(10) this　(11) const　(12) 缺省构造函数　(13) delete　(14) private，protected　(15) 非静态　(16) 内嵌类

6.2　选择题 (至少选一个，可以多选)

(1) 类定义描述的是_____。

a. 类的封装结构；

b. 与 struct (结构) 完全一样；

c. 创建时类的对象；

d. 与函数定义一样。

(2) 在类定义中，以下哪一个被指定为私有的数据或函数是可以访问的？_____。

a. 程序中的任何函数；

b. 仅当数据为常量或者知道该函数密码时；

c. 那个类的成员函数；

d. 那个类的公有成员或保护成员。

⑶ 构造函数不需要有返回类型，是因为_____。

a. 构造函数不是函数，所以不需要返回类型；

b. 构造函数使该类的对象初始化，所以它不需要有返回类型；

c. 构造函数可以被重载，所以不需要有返回类型；

d. 构造函数是系统自动调用的，没有程序能接收它的返回值，返回值没有任何意义。

⑷ 点（.）运算符（或成员访问运算符），从左至右的顺序连接以下两个实体是_____。

a. 类的对象和类；

b. 类和它的成员；

c. 类的成员和类的对象；

d. 类的对象和类的成员。

⑸ 成员函数总可以访问_____的数据。

a. 它所属的对象；

b. 它所属的类；

c. 它所属的任何类的任何对象；

d. 它所属的类的公共部分。

⑹ 类有很大的作用是因为它_____。

a. 在没有任何使用价值时从内存中清除掉；

b. 允许对其他的类隐藏数据；

c. 可以整合实体对象的方方面面；

d. 能够十分接近地为现实世界中的对象建模。

⑺ 拷贝构造函数与赋值运算符（=）的不同是_____。

a. 拷贝构造函数是复制，赋值运算符是赋予；

b. 拷贝构造函数会创建一个新的对象，而赋值运算符作用于已存的对象；

c. 拷贝构造函数与赋值运算符本质上是相同的；

d. 拷贝构造函数是复制，但不会创建一个新的对象，而赋值运算符会创建一个新的对象。

⑻ 对调用它的对象来说，常量成员函数_____。

a. 能够完全改变常量成员函数和非常量成员函数；

b. 只能改变非常量成员函数的数据；

c. 只能改变常量成员函数的数据；

d. 既不能改变常量成员数据也不能改变非常量成员的数据。

答：⑴ c ⑵ c ⑶ d ⑷ d ⑸ a ⑹ b，c，d ⑺ a，b ⑻ d

6.3 找出下面程序片段中的错误并解释其原因，设法加以更正。

⑴ 假定在 Time 类中说明为下述原型：

```
void  ~Time(int);
```

⑵ 下面是 Time 类的部分定义：

```
class Time {
  public:
    //函数原型
  private:
    int hour = 0;
    int minute = 0;
    int second = 0;
};
```

⑶ 假定下面的原型在 Employee 类中声明：

```
int Employee(const char *, const char *);
```

⑷ 下面是 Employee 类的新的定义：

```
class Employee {
    public:
      Employee(int y = 10) {
            data = y;
      }
      int getIncrementedData() const {
            return ++data;
      }
      static int getCount() {
        cout << "Data is" << data << endl;
        return count;
      }
    private:
        int data;
        static int count;
      };
```

⑸

```
char *string;
string = new char [20];
free(string);
```

答：⑴ 错误：析构函数是不允许有返回值，也不允许有参数。

更正：去掉返回类型 void 和声明中参数 int。

⑵ 错误：在类定义中成员不能显示地初始化。

更正：在类定义中去掉显示初始化，而在构造函数中初始化数据成员。

⑶ 错误：构造函数不允许有返回值类型。

更正：在说明中去掉返回类型 int。

⑷ 错误 1：在 employee 类的定义中有两个错误：第一，错误在 getIncrementedData () 函数中。该函数说明为 const，但它却修饰了对象。

更正 1：为此，应该把 const 从这个函数定义中删除掉。

错误 2：第二个错误产生在 getCount () 函数中，该函数说明为 static，所以就不允许访问任何非静态的类成员。

更正 2：为此，必须把 getCount () 函数定义中的输出行去掉。

⑸ 错误：用 new 进行动态内存分配，而用 C 标准库函数 free 进行回收内存是不合适的。

更正：应用 C++ 的 delete 操作符来回收内存。注意千万不要把 C 风格的内存分配与 C++ 的 new 和 delete 相混淆。

6.4 **试设计一个 Rabbit (兔子) 类。该类有 age，sex，speed 等属性以及对这些属性的操作的方法，然后实现并测试 Rabbit 类。**

答：

```
//Design Rabbit class
#include <iostream>
using namespace std;
class Rabbit {
    public:
      Rabbit(int initialAge=0,char initialSex='F',int initialSpeed=10);
      ~Rabbit();
      int GetAge() {return itsAge;}                    //内联函数
      int SetAge()(int age) {itsAge=age;}              //内联函数
      char GetSex() {return itsSex;}                   //内联函数
      char SetSex()(char Sex) {itsSex=Sex;}            //内联函数
      int GetSpeed() {return itsSpeed;}                //内联函数
      int SetSpeed(int Speed) {itsSpeed=Speed;}        //内联函数
    private:
      int itsAge,itsSpeed;
      char itsSex;
};
Rabbit::Rabbit(int initialAge=0,char initialSex='F',
                    int initialSpeed=10) {
      itsAge=initialAge;
      itsSex=initialSex;
      itsSpeed=initialSpeed;
}
Rabbit::~Rabbit() {}
int main() {
    Rabbit Mary(3,'F',12);
    cout<<"Mary is a Rabbit who is";
    cout<<Mary.GetAge()<<"years old,";
    cout<<Mary.GetSex()<<"Sex is and";
    cout<<Mary.GetSpeed()<<"m/s speed.";
```

```
    Mary.SetAge(6);
    Mary.SetSpeed(16);
    cout << "Now Mary is";
    cout << Mary.GetAge() << "years old and";
    cout << Mary.GetSpeed() << "m/s speed";
    return 0;
}
```

程序输出为：

```
Mary is a Rabbit who is 3 years old, sex is F and 12 m/s speed.
Now Mary is 6 years old and 16 m/s speed.
```

6.5 **试定义一个古树 (ancintTree) 类。此树首先包含一个成员** ages，**一个可将古树年龄不断增长的成员函数** grow (int years)，**并要求成员函数** age () **显示** ancintTree **对象的** ages **值。**

答：

```
//a ancint Tree is growing
#include < iostream >
using namespace std;
class ancintTree {
    int ages;
public:
    ancintTree(int n = 0);
    ~ancintTree();
    void grow(int years);
    void age();
};

ancintTree::ancintTree(int n) {
        ages = n;
}
ancintTree::~ancintTree() {
        age();
}
void ancintTree::grow(int years) {
        ages += years;       //古树增长的年龄
}
void ancintTree::age() {
    cout << "这棵古树的树龄为" << ages << endl;
}
void main() {
      ancintTree atree(98);
      atree.age();
```

```
        atree.grow(6);
    }
```

程序输出为

这棵古树的树龄为 98

这棵古树的树龄为 104

6.6 试创建一个模拟基本数据类型 float 的部分功能的类，我们姑且把这个类称为 FLOAT。该类中惟一的数据是一个 float 类型的变量，还包含一些成员函数，比如用来给一个 FLOAT 赋初值 0.0，将它初始化为 float 类型的数值，显示它就像显示基本数据类型 float 那样的数值，并且将两个 FLOAT 值进行相加。

答：

```
//用一个类去模拟浮点数据类型 float
#include <iostream>
using namespace std;
class FLOAT{     //与 float 类型不一样
    private:
       float i;
    public:
    FLOAT(){     //创建一个 FLOAT
        i=0.0;
    }
FLOAT(float ii){     //创建并初始化一个 FLOAT
        i=ii;
    }
    void add(FLOAT i2,FLOAT i3){     //两个 FLOAT 相加
        i=i2.i+i3.i;
    }
    void display(){     //显示一个 FLOAT
        cout<<i;
    }
int main(){
    FLOAT FLOAT1(9.0);                  //创建并初始化一个 FLOAT
    FLOAT FLOAT2(13.0);                 //又创建并初始化一个 FLOAT
    FLOAT FLOAT3;                       //创建第 3 个 FLOAT
    FLOAT3.add(FLOAT1,FLOAT2);          //两个 FLOAT 相加
    cout<<"\n FLOAT3=";
    FLOAT3.display();                   //显示结果
    cout<<endl;
    return 0;
}
```

第 7 章　继　　承

7.1　填空题

⑴ 如果类 A 继承了类 B，那么类 A 被称为_____类，而类 B 又被称为_____类。

⑵ C++ 提供了_____机制，允许一个导出类可以继承多个基类，甚至这些基类是互不相关的。

⑶ 继承能够缩短开发时间，并能充分地使用以前千锤百炼的高质量的软件。我们把这种效果称为_____。

⑷ 当从一个带有 public (公有) 继承的基类中导出一个类时，这个基类的 public 成员就成为该导出类的_____成员，而这个基类的 protected 成员就成为了该导出类的_____成员。

⑸ 当从一个带有 protected (保护) 继承的基类中导出一个类时，这个基类的 public 成员就成为了该导出类的_____成员，而这个基类的 protected 成员就成为了该导出类的_____成员。

⑹ _____类的对象能够被当作相应的_____类的对象来处理。

⑺ 类与类之间的“his _ a”关系表示为_____，而类与类之间的“is _ a”关系又表示为_____。

⑻ 要能够被导出类的成员函数访问，基类中的数据成员必须是 public 或_____。

⑼ 为了把基类的指针转换成导出类的指针，因为编译器认为这是一种危险的操作，所以就必须使用_____。

⑽ 组装是一个_____的_____形式。

答：⑴ 导出，基　⑵ 多重继承　⑶ 软件可重用性　⑷ public，protected

⑸ protected，protected　⑹ 导出，基　⑺ 组装，继承　⑻ protected

⑼ cast　⑽ 更强大，聚合关系

7.2　选择题 (至少选一个，可以多选)

⑴ 继承方式有三种：public (公有继承)，protected (保护继承)，private (私有继承)。它们之间的差别是_____。

a. 除了公有继承外，基类的保护继承和私有继承的成员都不可以访问；

b. 对公有继承而言，基类的 public 和 protected 成员在导出类中可以访问，但基类的 private 成员不可以访问；

c. 对私有继承而言，基类的 public 和 protected 成员都以 private 成员身份出现在导出类中，但基类的 private 成员不可以访问；

d. 对保护继承而言，基类的 public 和 protected 成员都以 protected 成员身份出现在导出类中，但基类的成员不可以访问；

e. 对私有和保护继承而言，基类的 public 和 protected 成员都以 public 成员身份出现在导出类中，但基类的成员不可以访问。

⑵ 导出类构造函数执行次序是_____。

a. 调用基类构造函数，然后调用导出类对象的构造函数，再执行导出类的构造函数体中的内容；

b. 调用基类构造函数，然后调用成员对象的构造函数，再执行导出类的构造函数体中的内容；

c. 调用基类构造函数，然后调用导出类的构造函数，再执行导出类的构造函数体中的内容；

d. 调用基类构造函数，然后调用基类成员对象的构造函数，再执行导出类的构造函数体中的内容。

⑶ 继承是_____的方法。

a. 将特殊的类变成通用的类；

b. 把通用的参数传送给特殊的类的对象；

c. 将通用的类变成特殊的类；

d. 改进通用类的数据隐藏和封装；

e. 将已有的类添加新的特性，但不重写它们。

⑷ 继承的优点是_____。

a. 扩大类的使用范围，更便于使用类库；

b. 避免重写程序代码，提供有用的概念框架；

c. 把类组织成有条理的特化层次结构；

d. 通用继承的自然选择和重写使类进一步拓展。

⑸ 作用域限定运算符通常是_____。

a. 指定特定的类；

b. 指明从哪一个基类中导出来的；

c. 在某些成员函数中限定静态变量的可视范围；

d. 解析二义性。

⑹ 类层次_____。

a. 描述类的“隶属”关系；

b. 描述类的“组成”关系；

c. 描述类的“相同”关系；

d. 把描述“相同”的类关系组成系统族图谱。

⑺ 聚合是_____。

a. 更强大的继承关系；

b. 更强大的泛化关系；

c. 更强大的组装关系；

d. 一种“隶属”关系。

⑻ 多重继承是_____。

a. 多个单一继承的叠加；

b. 导出类有多个直接基类；

c. 多个导出类有惟一的基类；

d. 每个导出类最多只有一个直接基类，但它可以有多个间接基类。

答：(1) b，c，d (2) b (3) c，e (4) a，b，c (5) a，d (6) b (7) d (8) b

7.3 假定基类和导出类有同名的成员函数。如果不使用作用域限定运算符，则哪个成员函数会被导出类调用？

答：导出类中的那个。

7.4 试创建一个哺乳动物 Mammal 类，然后从哺乳动物类中导出兔子 Rabbit 类，再定义一个 Rabbit 类的对象。请读者编制出基类与导出类的构造函数与析构函数的调用次序的 C++ 源程序，并用 main () 函数来观察输出情况。

答：//基类与导出类关系源代码

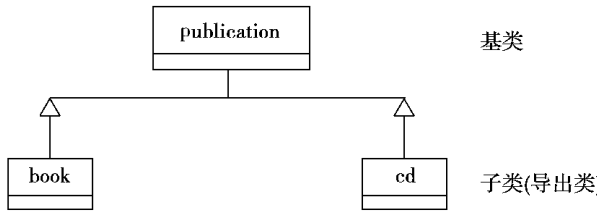
```
#include <iostream>
using namespace std;
enum iColor{WHITE,GREY};
class Mammal{
  public:
    Mammal();                                  //无参构造函数
    ~Mammal();                                 //析构函数
    int getAge()const{return itsAge;}          //访问数据成员
    void setAge(int age){itsAge = age;}
    int getSpeed()const{return itsSpeed;}
    void setSpeed(int speed){itsSpeed = speed;}
    //其他函数
    void Reat()const{coout << "Mammal is eating foods! \n";}
  protected:
    int itsAge;
    int itsSpeed;
};
class Rabbit:public Mammal{
  public:
    Rabbit();
    ~Rabbit();
    iColor getColor()const{return itsColor;}
    void setColor (icolor color){itsColor = color;}
    void eatVegetable(){cout << "Rabbit is eating vegetable...\n";}
  private:
    iColor itsColor;
};
Mammal::Mammal():itsAge(1),itsSpeed(5){
    cout << "Mammal constructor...\n";
}
```

```
Mammal::~Mammal(){
    cout<<"Mammal destructor...\n";
}
Rabbit::~Rabbit(){
    cout<<"Rabbit destructor...\n";
}
int main(){
  Rabbit Mary;
  Mary Reat();
  Mary eatVegetable();
  cout<<"Mary is"<<Mary.getAge()<<"years old.\n";
  return  0;
}
```

程序运行结果如下：

```
Mammal constructor...
Rabbit constructor...
Mammal is eating foods!
Rabbit is eating vegetable...
Mary is 1 years old.
Rabbit destructor...
Mammal destructor...
```

7.5 **我国某一出版社有两类出版物：图书和音像制品（比如光碟）。试创建一个** publication **类来存储出版物的标题和价格，其属性分别为字符串类型和浮点数（**float**）类型。然后从** publication **类导出（派生）两个类：**book **类和** cd **类。**book **类含有表示页数的属性，为整数类型；**cd **类含有以分钟为单位的播放时间属性，为** float **类型。以上三个类关系如图题** 7-1 **所示（**OMT **对象图，见第** 12 **章面向对象建模）。**



图题 7-1　**基类与子类关系**

这三个类都具有两个函数：

- getData () **函数　用来通过键盘从用户那里获得数据。**
- putData () **函数　用来显示数据。**

请读者编写一个通过创建 book **类和** cd **类的实例来测试这两个类的** C++ **程序，然后由用户使用** getData () **函数向其中添加数据，并用** putData () **函数将这些数据显示出来。**

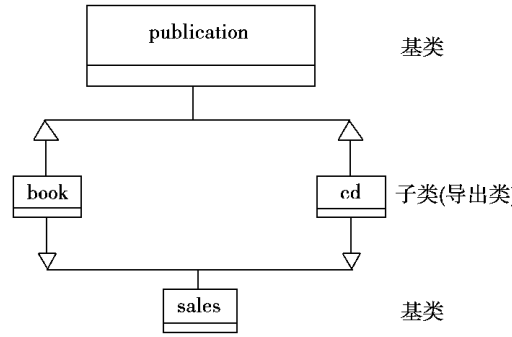
答：

```
//publication class and two derived class (book,cd)
#include < iostream >
#include < string >
using namespace std;
class publication{  //基类
  private:
    string title;
    float price;
  public:
    void getData(){
      cout << "Please enter title:";
      cin >> title;
      cout << "Please enter price:";
      cin >> price;
    }
    void putData()const{
      cout << "\n Title:" << title;
      cout << "\n Price:" << price;
    }
};
class book:private publication {  //导出类
  private:
    int pages;
  public:
    void getData(){
      publication::getData();
      cout << "Please enter number of pages:";
      cin >> pages;
    }
    void putData()const{
      publication::putData();
      cout << "\n Pages:" << pages;
    }
};
class cd:private publication {  //导出类
    private:
      float time;
    public:
      void getData(){
```

```
        publication::getData();
        cout<<"Please enter playing time:";
        cin>>time;
      }
      void putData()const{
        publication::putData();
        cout<<"\n Playing time:"<<time;
      }
};
int main(){
  book bookone;          //定义 publication 类导出的两个类的对象
  cd cdone;
  bookone.getData();     //从上述两个对象中获取数据
  cdone.getData();
  bookone.putData();     //显示这两个对象的数据
  cdone.putData();
  cout<<endl;
  return 0;
}
```

7.6 **如果在 7.5 题基础上，再添加一个基类 sales，这时我们一共有了 4 个类：publication 类、book 类、cd 类和 sales 类。此外，我们还有一个存储三个 float 类型的数的数组，用来记录过去六个月以来特定出版物的销售额（包括用来从用户那里获得三个销售总额的 getData() 函数和显示销售数字的 getData() 函数）。**

试修改 book 类和 cd 类，使得这两个类通过 publication 和 sales 基类导出来，而 book 类和 cd 类的对象应该与它们自己其他的数据一起输入/输出销售额的数据。并编制 main() 函数创建一个 book 对象和一个 cd 对象，来测试它们的 I/O 能力。



图解 7-1 publication 类、book 类、cd 类及 sales 类的关系

答：按 OMT 对象图这 4 个类的关系图解 7-1 所示（多重继承）。

```
//publication 类的多重继承
#include<iostream>
#include<string>
using namespace std;
class publication{  //基类
  private:
```

```
    string title;
    float price;
  public:
    void getData(){
      cout<<"\n Please enter title:";
      cin>>title;
      cout<<"Please enter price:";
      cin>>price;
    }
    void putData()const{
      cout<<"\n Title:"<<title;
      cout<<"\n Price:"<<price;
    }
};
class sales {  //基类
  private:
    enum{MONTHS=6};
    float salesArray[MONTHS];
  public:
    void getData();
    void putData()const;
};
void sales::getData(){
    cout<<"Please sales for six months\n";
    for (int j=0;j<MONTHS;j++){
      cout<<"Month"<<j+1<<":";
      cin>>salesArray[j];
    }
}
void sales::putData()const{
  for(int j=0;j<MONTHS;j++){
    cout<<"\n Sales for month"<<j+1<<":";
    cout<<salesArray[j];
  }
}
class book:private publication,private sales{
  private:
    int pages;
  public:
```

```
    void getData(){
      publication::getData();
      cout << "Please enter number of pages:";
      cin >> pages;
      sales::getData();
    }
    void putData()const{
      publication::putData();
      cout << "\n Pages:" << pages;
      sales::putData();
    }
};
class cd:private publication,private sales{
  private:
    float time;
  public:
    void getData(){
      publication::getData();
      cout << "Please enter playing time:";
      cin >> time;
      sales::getData();
    }
    void putData()const{
      publication::putData();
      cout << "\n Playing time:" << time;
      sales::putData();
    }
};
int main(){                    //主函数
  book bookone;                //定义 publication 类导出的两个类的对象
  cd cdone;
  bookone.getData();           //从 publication 获得数据
  cdone.getData();
  bookone.putData();           //显示数据
  cdputData();
  cout << endl;
  return  0;
}
```

第 8 章　多态、虚拟函数和模板

8.1　填空题

⑴ 使用继承和多态能够对消除_____逻辑有帮助。

⑵ 纯虚拟函数是在定义中原型结束处用_____指定。

⑶ 如果一个类包含一个或更多个的纯虚拟函数，那么它就是一个_____。

⑷ 联编是指将函数调用与相应函数体代码彼此关联的过程。若此过程在程序开始运行前的编译时完成，则称之为_____联编。

⑸ 若在运行分解函数调用，则称之为_____联编。

⑹ 模板能够使我们用一个统一的代码段指定相关的（或重载的）函数的整个_____，这就称为_____；或者指定相关类的整个作用域，则称之为_____。

⑺ 所有函数模板都用_____关键字定义，接着在_____符号和_____符号范围内列出函数模板的模板形参表。

⑻ 因为用所有相同名字的函数模板产生的相应的函数，所以编译器可以使用_____方式来引用合适的函数。

⑼ 类模板也总是称为_____类型。

⑽ 就像非模板类的 static 数据成员一样，模板类的 static 数据成员也必须在_____作用域内初始化。

⑾ 模板概念的关键之处在于用_____替代了_____的名字。

⑿ 模板所创建的一个实际的函数，称之为_____函数。

答：⑴ switch　⑵ = 0　⑶ 抽象基类　⑷ 静态　⑸ 动态

⑹ 作用域，模板函数，模板类　⑺ templete，< ，>　⑻ 重载

⑼ 参数化　⑽ 文件　⑾ 固定数据类型，任意数据类型　⑿ 实例化

8.2　选择题（至少选一个，可以多选）

⑴ 虚拟函数允许_____。

a. 把不同类的对象将它们放在同一个数组中，该数组能够保存指向导出类的指针；

b. 创建永远不会被导出类访问的函数；

c. 把不同类的对象聚集在一起，以便它们能够很容易地被相同的函数代码访问；

d. 用相同的函数调用执行不同类对象的成员函数。

⑵ 纯虚拟函数是一个虚拟函数，而且_____。

a. 使所属的类变为抽象类；

b. 永远不会返回任何东西；

c. 使用在基类（父类）之中；

d. 不带任何参数。

⑶ 当_____抽象类是很有用的。

a. 没有类应该由它导出时；

b. 从一个导出类到另一个有多条路径时；
c. 它永远不想去创建一个类的对象实例时；
d. 为类层次提供一个接口；
e. 设法尽量推迟类的声明时。

(4) 友元函数可以用来_____。
a. 在类与类之间有规律的传递参数；
b. 允许访问哪些未经授权的源代码类；
c. 允许访问互不相关的类；
d. 拓展了重载运算符的功能。

(5) friend 关键字可以出现在_____。
a. 允许另一个类访问的类中；
b. 要求并希望访问另一个类的类中；
c. 类的私有 (private) 部分；
d. 类的公有 (public) 部分。

(6) 模板提供了简约的方法建立一系列的_____。
a. 变量；
b. 函数；
c. 类；
d. 程序；
e. 模块。

(7) 类模板_____。
a. 的设计是为了存储在不同的包容器中；
b. 使用了不同的数据类型；
c. 产生的对象都必须被惟一的标识；
d. 产生的类其成员函数个数亦不同。

(8) 虽然宏与函数模板有相同的效果，但函数模板与宏不同的是_____。
a. 宏不进行任何类型检测。宏中多个参数应该是同一类型，但编译器并不检测；
b. 宏的返回结果不确定，宏被限定为只能用一条语句表示的函数；
c. 宏在 C++ 语言中比函数模板使用的多；
d. C++ 编译器可以确定宏中参数不确定的类型。

(9) 静态函数_____。
a. 当对象删除时应当被调用；
b. 可以用类名和函数名来访问；
c. 与类中的单个对象紧密联系着；
d. 当一个哑对象 (dummy object) 必须创建时方可使用。

(10) 模板函数_____。
a. 并不是真正的函数，只是一种产生各种函数的模式或框架；
b. 是真正的函数，与函数的定义没有什么不同；
c. 并不是真正的函数，而是函数的原型；

d. 与面向对象程序设计（OOP）的观点一致，是产生多种类似的物体的框架。

答：(1) d　(2) a，c　(3) c，d　(4) c，d　(5) a，c，d　(6) b，c
(7) b　(8) a，b　(9) b　(10) a，d

8.3 判断题

试回答下面的句子是否正确，如判为错误，则要回答为什么？

(1) 一个函数模板的友元函数一定是一个模板函数。

(2) 如果几个模板类是用统一的 static 数据成员由一个单一类模板产生，那么该模板类的每个共享了该类模板的 static 数据成员的统一拷贝。

(3) 一个模板函数能够被用相同函数名的另外模板函数重载。

(4) 一个形参的名字只能够在模板定义的形参列表中使用一次。在模板定义中的形参名必须惟一。

(5) 作为用模板类型参数的关键字 class，特别地含义是“任何用户定义类的类型”。

(6) 模板类不能够嵌套。

答：(1) 错误。它可以是一个非模板函数。

(2) 错误。每个模板类将有一个该 static 数据成员的拷贝。

(3) 正确。

(4) 错误。在模板函数中形参名不是惟一的。

(5) 错误。在这种上下文（语文）中关键字 class 也允许内置类型的类型参数。

(6) 正确。

8.4 该题有两问：

(1) 试解释以下两个语句之间的不同之点：

```
person p1(p0);
person p1 = p0;
```

(2) 试编写一个可以返回一个参数两倍值的模板。

答：(1) 操作完全等价。

(2)
```
template < class T >
T times2(T arg){
  return arg * 2;
}
```

8.5 试编写一个类声明，使之在这个类中每一个类 rabbit 的成员函数都是友元函数。

答：
```
friend class rabbit;
```
或者
```
friend rabbit;
```

8.6 试编写计算雇员工资系统的 C++ 程序。要求用虚拟函数和多态性来实现。
（提示：基类用 Employee 类，它的导出类有 4 个，分别是 Boss（老板）类、Commission-Worker（佣金工）类、PieceWorker（计件工）类和 HourlyWorker（小时工）类）。

答：
```
//抽象基类 Employee 的定义(employee.h)
#ifndef EMPLOYEE_H
#define EMPLOYEE_H
```

```
class Employee{
  public:
    Employee(const char * first,const char * last);     //构造函数
    ~Employee();                                        //析构函数
    const char * getFirstName()const;
    const char * getLastName()const;
    //Employee 的抽象基类由纯虚拟函数构成
    virtual float earings()const =0;                    //纯虚拟函数
    virtual void print()const =0;                       //纯虚拟函数
  private:
    char * firstName;
    char * lastName;
};
#endif
//抽象基类 Employee 成员函数的定义(EMPLOYEE.CPP)
//注意 没给出纯虚拟函数的定义
#include <iostream.h>
#include <string.h>
#include <assert.h>
#include"employee.h"
//对名字(firstname)和姓(lastname)为构造函数动态地分配空间,
//并用 strcpy 把名字和姓拷贝到对象中去
Employee::Employee(const char * first,const char * last){
    firstName = new char [strlen(first) +1];
    assert(firstName!=0);  //测试
    strcpy (firstName,first);
    lastName = new char [strlen(last) +1];
    assert(lastName !=0);  //测试
    strcpy(lastName,last);
}
//动态地撤销已分配的内存
Employee:: ~Employee(){
    delete [] firstName;
    delete [] lastName;
}
//返回一个指向名字(firstname)的指针
const char * Employee::getFirstName()const{
//用 const 可以避免调用者擅自修改私有数据,在析构函数撤销动态内存(避免使用未
  定义指针)之前,
```

```
//调用者应该返回串的拷贝
return firstName;  //调用者必须撤销内存
}
//返回一个指向姓(last name)的指针
const char *Employee::getlastName()const{
  return lastName;
}
//从抽象类 Employee 导出(派生)Boss 类(boss.h)
#ifndef BOSS_H
#define BOSS_H
#include "employee.h"
class Boss:public Employee{
  public:
    Boss(const char*,const char*,float=0.0);
    void setWeeklySalary(float);
    virtual float earnings()const;
    virtual void print()const;
  private:
    float weeklySalary;
};
#endif
//Boss 类的成员函数定义(boss.cpp)
#include<iostream.h>
#include"boss.h"
//Boss 类的构造函数
Boss::Boss(const char *first,const char*last,floats)
    :Employee(first,last){  //调用基类构造函数
  setWeeklySalary(s);
}
//设定 Boss 的工资
void Boss::setWeeklySalary(floats){
    weeklySalary=s>0? s:0;
}
//获得 Boss 的薪金
float Boss::earings()const{
    return weeklySalary;
}
//打印 Boss 的名字
```

```
void Boss::print()const{
  cout << endl << "Boss:" << getFirstName()
       << '' << getLastName();
}
//由 Employee 导出 CommissionWorker 类(comm.h)
#ifndef COMMIS_H
#define COMMIS_H
#include"employee.h"
class CommissionWorker:public Employee{
  public:
    CommissionWorker (const char *,const char *,float=0.0,float=0.0,
    unsigned=0);
  void setSalary(float);
  void setCommission(float);
  void setQuantity(unsigned);
  virtual float earing()const;
  virtual void print()const;
private:
  float salary;              //每个星期的基本工资
  float commission;          //每项销售佣金的总和
  unsigned quantity;         //每星期销售项目的总和
};
#endif
//佣金工 CommissionWorker 类的成员函数定义(commis.cpp)
#include<iostream.h>
#include"commis.h"
//CommissionWorker 类的构造函数
CommissionWorker::CommissionWorker(const char *first,const char *last,
                    float s,float c,unsigned q)
       :Employee(first,last){  //调用基类构造函数
       salary=s>0? s:0;
       commission=c>0? c:0;
       quantity=q>0? q:0;
}
//设定佣金工每星期的基本工资
void CommissionWorker::setSalary(float s){
       salary=s>0? s:0;
}
//设定佣金工的佣金
```

```
void CommissionWorker::setCommission(float c){
        commission = c > 0? c:0;
}
//设定佣金工销售量
void CommissionWorker::setQuantity(unsigned q){
        quantity = q > 0? q:0;
}
//确定佣金工所挣得的酬金
float CommissionWorker::earings()const{
    return salary + commission * quantity;
}
//打印佣金工的名字
void CommissionWorker::print()const{
  cout << endl << "Commission Worker:" << getFirstName()
     << '' << getLastName();
}
//从 Employee 导出 PieceWorker 类(piece.h)
#ifndef PIECE_H
#define PIECE_H
#include "employee.h"
class PieceWorker:public Employee{
  public:
    PieceWorker(const char *,const char *,float =0.0,unsigned =0);
    void setWage(float);
    void setQuantity(unsigned);
    virtual float earnings()const;
    virtual void print()const;
  private:
    float wagePerPiece;       //每个计件酬金的输出
    unsigned quantity;        //一星期的输出量
};
#endif
//计件工 PieceWorker 类的成员函数定义(piece.cpp)
#include <iostream.h>
#include"piece.h"
//PieceWorker 类的构造函数
PieceWorker::PieceWorker(const char * first,const char * last,float w,
                unsigned q)
```

```
        :Employee(first,last){  //调用基类构造函数
        wagePerPiece = w > 0? w:0;
        quantity = q > 0? q:0;
}
//设定酬金
void PieceWorker::setWage(float w){
    wagePerPiece = w > 0? w:0;
}
//设定输出的项目件数量
void PieceWorker::earings()const{
    return quantity * wagePerPiece;
}
//打印计件工的名字
void PieceWorker::print()const{
  cout << endl << "Piece Worker:" << getFirstName()
 << '' << getLastName();
}
//小时工 HourlyWorker 类的定义(hourly.h)
#ifndef HOURLY_H
#define HOURLY_H
#include"employee.h"
class HourlyWorker:public Employee{
  public:
    HourlyWorker(const char *,const char *,float = 0.0,float = 0.0);
    void setWage(float);
    void setHours(float);
    virtual float earnings()const;
    virtual void print()const;
  private:
    float wage;              //每小时所挣的酬金
    float hours;             //每个星期工作的小时数
};
#endif
//小时工 HourlyWorker 类成员函数的定义(hourly.cpp)
#include < iostream.h >
#include"hourly.h"
//HourlyWorker 类的构造函数
HourlyWorker::HourlyWorker(const char * first,const char * last,
float w,float h)
```

```
        :Employee(first,last){  //调用基类构造函数
        wage = w > 0? w:0;
        hours = h > 0&&h < 168? h:0;
}
//设定小时工酬金
void HourlyWorker::setWage(float w){
        wage = w > 0? w:0;
}
//设定工作的小时数
void HourlyWorker::setHours(float h){
        hours = h >= 0&& h < 168? h:0;
}
//挣得小时工的工资
float HourlyWorker::earnings()const{
    return wage * hours;
}
//打印小时工的名字
void HourlyWorker::print()const{
    cout << endl << "Hourly worker:" << getFirstName()
      << '' << getLastName();
}
//雇员工资系统主控程序(导出 Employee 的层次结构)
#include < iostream.h >
#include < iomanip.h >
#include"employee.h"
#include"boss.h"
#include"commis.h"
#include"piece.h"
#include"hourly.h"
int main(){
  //设定输出格式
  cout << setiosflag(ios::showpoint) << setprecision(2);
  Employee * ptr;                                    //基类指针
  Boss b("Yankai","Wan",800.00);
  ptr = &b;                                          //基类指针指向导出类对象
  ptr -> print();                                    //动态联编
  cout << "earned RMB" << ptr -> earnings();         //动态联编
  b.print();                                         //静态联编
```

```
  cout << "earned RMB" << b.earnings();          //静态联编
  CommissionWorker C("Liqin","Qiao",200.00,3.0,150);
  ptr = &c;                                     //基类指针指向导出类对象
  ptr -> print();                               //动态联编
  cout << "earned RMB << ptr -> earnings();     //动态联编
  c.Print();                                    //静态联编
  cout << "earned RMB << c.earnings();          //静态联编
  PieceWorker p("Baolin","Li",2.5,200);
  Ptr = &p;                                     //基类指针指向导出类对象
  ptr -> print();                               //动态联编
  cout << "earned RMB" << ptr -> earnings();    //动态联编
  p.print();                                    //静态联编
  cout << "earned RMB" << p.earnings();         //静态联编
  HourlyWorker h("Tennis","Moon",13.75,40);
  ptr = &h;                                     //基类指针指向导出类对象
  ptr -> print();                               //动态联编
  cout << "earned RMB" << ptr -> earnings();    //动态联编
  h.print();                                    //静态联编
  cout << "earned RMB" << h.earnings();         //静态联编
  cout << endl;
  return 0;
}
```

运行结果为：

```
            Boss:YanKai Wan earned RMB 800.00
            Boss:YanKai Wan earned RMB 800.00
Commission worker:Liqin Qiao earned RMB 650.00
Commission worker:Liqin Qiao earned RMB 650.00
      Piece Worker:Baolin Li earned RMB 500.00
      Piece Worker:Baolin Li earned RMB 500.00
   Hourly Worker:Tennis Moon earned RMB 550.00
   Hourly Worker;Tennis Moon earned RMB 550.00
```

简单解释主控程序

主控程序从声明基类指针 ptr **(是** Employee * **类型) 开始。在** main () **中顺序的三个代码段都是类似的，所以我们只解释第一个** Boss **对象代码段就可以了。**

```
Boss.b("YanKai","Wan",800.00);
```

实例化 Boss **类的导出类对象** b**，并提供了不同的构造函数参数，包括名字 (**first name**)、姓 (**last name**) 和每周固定工资。**

```
ptr = &b;  //基类指针指向导出类对象
```

把基类指针 ptr **置为导出类对象** b **的地址，这是多态性行为。**

```
ptr->print();  //动态联编
```
用 ptr 来引用指向该对象的 print() 成员函数。因为 print() 已经在基类中声明为虚拟函数，所以系统引用导出类对象 print() 函数，这又是多态性行为。这个函数调用是一个动态联编的例证（函数通过基类指针来引用），所以引用什么样的函数直到执行时方能确定。

```
cout<<"earned RMB"<<ptr->earnings();  //动态联编
```
用 ptr 来引用指向该对象的 earnings() 成员函数，这与 print() 成员函数是一样的，也是动态联编的一个例证。

```
b.print();  //静态联编
```
用“.”成员选择操作符对 Boss 对象 b 引用成员函数 print()，这是静态联编的一个例证。因为在编译时就可以知道调用这个函数的该对象类型。

```
cout<<"earned RMB"<<b.earnings();  //静态联编
```
也是显式引用（用“.”操作符）Boss 的对象 b 的成员函数 earnings()，这也是静态联编的一个例证。

第 9 章　运算符重载

9.1　填空题

⑴ 假定 a 和 b 是整型变量，则用 a + b 表示它们之和；又假定 c 和 d 是浮点型变量，我们用 c + d 表示它们之和。在此，所用的两个‘ + ’运算符清晰地表达了不同的目的。我们把‘ + ’称作_____。

⑵ _____关键字引入了重载运算符的函数定义。

⑶ 为在类对象上使用重载运算符，它们就必须在除了_____运算符和_____运算符之外进行重载。

⑷ _____运算符和_____运算符不能用重载运算符加以改变。

⑸ 重载流运算符通常做为一个类的_____函数定义。

⑹ 大部分流类的基类是_____；流插入运算符的符号是_____。

⑺ ostream 成员函数_____用来执行无格式化输出。

⑻ 与系统的标准设备相对应的四种对象是_____、_____、_____和_____。

⑼ _____头文件包括了用户控制文件处理信息；_____头文件在程序中将 C 风格和 C++ 风格的 I/O 混合起来。

⑽ 使用前缀形式时，与后缀形式相比，重载的 ++ 运算符有哪些地方不同：_____。

⑾ C++ 中输入/输出产生字节的_____。

⑿ C++ 中不能被重载的运算符有_____、_____、_____、_____和_____。

答：⑴ 运算符重载　⑵ operator　⑶ 赋值（=），地址（&）　⑷ 前置，结合

⑸ friend　⑹ ios，<<　⑺ endl　⑻ cin，cout，cerr，clog

⑼ fstream，stdiostream　⑽ 它与没有重载的 ++ 运算符一样，在使用变量之前将它递增

⑾ 流（stream）　⑿ ．，．*，::，?:，sizeof

9.2　选择题（至少选一个，可以多选）

⑴ 运算符重载_____。

a. 可使 C++ 运算符能够使用对象；

b. 赋予超出 C++ 运算符本身的处理能力；

c. 给已有的 C++ 运算符赋予新的定义；

d. 创造新的 C++ 语言从未有的运算符。

⑵ 重载算术赋值运算符时，其结果_____。

a. 存入运算符右方的对象中；

b. 存入运算符左方的对象中；

c. 返回一个对象的临时变量；

d. 存入运算符所属的对象中。

⑶ 若把用户自定义数据类型转换为基本数据类型的话，应最有可能会使用_____。

a. C++ 内置的转换运算符；
b. 重载的‘ = ’运算符；
c. 这个类中的转换运算符；
d. 转换构造函数（即一个参数的构造函数）。

⑷ 若把基本数据类型转换为用户自定义数据类型，应最有可能会使用_____。
a. C++ 内置的转换运算符；
b. 重载的‘ = ’运算符；
c. 这个类中的转换运算符；
d. 转换构造函数（即一个参数的构造函数）。

⑸ C++ 中流是_____。
a. 函数与函数之间的控制流；
b. 文件与文件之间的控制流；
c. 从一处到另一处的数据字节流；
d. 与特定的类有关联。

⑹ 我们可以用插入运算符 << 输出文本到类 ofstream 的对象，这是因为_____。
a. 类 ofstream 是流；
b. 插入运算符可以自行处理所有的类；
c. 插入运算符在类 ofstream 中得到重载；
d. “连接”到标准输出 cout 上。

答： ⑴ a，c ⑵ b，d ⑶ c ⑷ d ⑸ c，d ⑹ c

9.3 试解释在 C++ 中运算符 << 和运算符 >> 的多种含义。

答： 运算符 << 的含义：它即表示左移运算符、流输出的重载，又表示作为流 - 插入运算符来引用，这依赖于上下文。
运算符 >> 的含义：它既表示右移运算符、流输入的重载，又表示作为流 - 析取运算符来引用，这依赖于上下文。

9.4 试写出下列各语句的输出结果：

⑴
```
cout << "12345" << endl;
cout.width(5);
cout.fill('*');
cout <<123 << endl <<123;
```

⑵
```
cout << setw(10) << setfill('$') <<10000;
```

⑶
```
cout << setw(8) << setprecision(3) <<1024.987654;
```

⑷
```
cout << setiosflags(ios::showbase) << oct << 99
     << endl << hex << 99;
```

⑸
```
cout <<100000 <<endl
     << setiosflags(ios::showpos) <<100000
```

⑹
```
cout << setw(10) << setprecision(2) <<
     << setiosflags(ios::scientific) <<44.93738;
```

答： ⑴ 1 2 3 4 5
＊＊1 2 3
1 2 3
⑵ $ $ $ $ $10000
⑶ 1024. 988
⑷ 0143
0x63
⑸ 100000
+100000
⑹ 4. 45e + 02

9. 5 **试编写对** POINT **类重载** ++、 -- **运算符的** C++ **程序。**

答：
```
#include < iostream >
using namespace std;
class POINT{
  public:
    POINT&operator ++ ();
    POINT operator ++ (int);
    POINT&operator -- ();
    POINT operator -- (int);
    POINT(){_ x = _ y = 0;}
    int x(){return _ x;}
    int y(){return _ y;}
  private:
    int _ x,_ y;
};
POINT&POINT::operator ++ (){
      _ x ++;
      _ y ++;
      return * this;
}
POINT POINT::operator ++ (int){
      POINT temp = * this;
       ++ * this;
      return temp;
}
POINT&POINT::operator -- (){
      _ x --;
      _ y --;
```

```
        return *this;
    }
    POINT POINT::operator--(int){
        POINT temp=*this;
        --*this;
        return temp;
    }
    void main(){
        POINT X;
        cout<<"X 的值为:"<<X.x()<<","<<X.y()<<endl;
        X++;
        cout<<"X 的值为:"<<X.x()<<","<<X.y()<<endl;
        ++X;
        cout<<"X 的值为:"<<X.x()<<","<<X.y()<<endl;
        X--;
        cout<<"X 的值为:"<<X.x()<<","<<X.y()<<endl;
        --X;
        cout<<"X 的值为:"<<X.x()<<","<<X.y()<<endl;
    }
```

程序运行结果为：

X 的值为：0，0
X 的值为：1，1
X 的值为：2，2
X 的值为：1，1
X 的值为：0，0

9.6 **试编写有如下要求的** C++ **程序：**

⑴ 定义一个有成员变量 XPoint，YPoint **的** POINT **类；**

⑵ 为 POINT **类定义的友元函数实现重载运算符‘** + **’。**

答：

```
#include<iostream>
using namespace std;
class POINT{
  public:
    POINT(){
      XPoint=0;
      YPoint=0;
   }
    POINT(unsigned x,unsigned y){
    XPoint=x;
    YPoint=y;
```

```
  }
    unsigned x(){return XPoint;}
    unsigned y(){return YPoint;}
    void POINT(){
      cout << "POINT(" << XPoint << "," << YPoint << ")"
        << endl;
    }
    friend POINT operator+(POINT&pt,int NOffset);
    friend POINT operator+(int NOffset,POINT&pt);
  private:
    unsigned XPoint;
    unsigned YPoint;
};
POINT operator+(POINT&pt,int NOffset){
    POINT ptTemp = pt;
    ptTemp.XPoint += NOffset;
    ptTemp.YPoint += NOffset;
    return ptTemp;
}
POINT operator+(int NOffset,POINT&pt){
    POINT ptTemp = pt;
    ptTemp.XPoint += NOffset;
    ptTemp.YPoint += NOffset;
    return ptTemp;
}
void main(){
    POINT Pt(5,5);
    pt.Print();
    pt = pt + 5;  //POINT + int
    pt.Print();
    pt = 10 + pt;  //int + POINT
    pt.Print();
}
```

运行结果：

POINT(5，5)

POINT(10，10)

POINT(20，20)

第 10 章　包 容 器 类

10.1　填空题

⑴ C++ 提供了两种表示选择：一种表示是_____，另一种表示是_____。

⑵ STL 中顺序包容器有_____、_____和_____。

⑶ STL 中关联包容器有_____和_____。

⑷ 迭代器_____包容器中指定的元素。

⑸ 算法的范围通常由两个_____指定。

⑹ 使用 new 操作符比使用数组更_____空间。

答：⑴ 基于数组，基于类　⑵ 向量 (vector)，列表 (list)，双端队列 (deque)

⑶ 集合 (set)，映射 (map)　⑷ 指向　⑸ 迭代器　⑹ 节省

10.2　选择题 (至少选一个，可以多选)

⑴ STL 可用于_____。

a. 以某种方法存储元素以便快速访问；

b. 编译或编辑 C++ 源代码；

c. 在内存中组织对象的一种方式；

d. 保存比如像类 student 那样的对象。

⑵ STL 算法是_____。

a. 对成员函数执行的独立操作；

b. 对包容器执行操作的独立函数；

c. 对适配的包容器类的友元函数执行的操作；

d. 对适配的包容器类的成员函数执行的操作。

⑶ 如希望 vector 成为一个合适的包容器，那就应该_____。

a. 在 vector 中任意位置上插入许多新元素；

b. 经常在包容器的头部插入许多新元素；

c. 给定一个索引值，以便能迅速地访问所符合的元素；

d. 给定一个元素的键值，以便能迅速地访问所符合的元素。

⑷ 在双端队列 deque 中，_____。

a. 其元素可以在任何位置上迅速地插入、删除；

b. 其元素可以在任何位置上速度较慢地插入、删除；

c. 其元素可以在任何一端迅速地插入、删除；

d. 其元素可以在任何一端速度较慢地插入、删除。

⑸ 在关联容器中，_____。

a. 值按排序后的次序存储；

b. 键按排序后的次序存储；

c. 按照字母大小或数值大小的次序排序；

d. 必须用排序函数 sort () 重新排列内容次序。

(6) 倘若在包容器中存储的是对象的指针，而不是对象，则_____。

a. 包容器中实现存储时没有必要复制对象；

b. 只有关联包容器可以使用指针；

c. 不可以把对象的指针作为键对象排序；

d. 包容器所需的内存更少。

答：(1) a，c，d (2) b (3) c (4) b，c (5) b (6) a，d

10.3 试编写一个语句，执行 rabbit 类的其中一个对象的成员函数 eating ()，而这个对象是数组 Arrayrabbit 的第 19 个元素。

答：Arrayrabbit [18]. eating () ；

10.4 假定 pt 是指向类 Employee 的对象的指针，试编写一个语句执行该对象的成员函数 salary ()。

答：

pt - > salary () ；

10.5 假定我们有一类 student，创建一个保存 student 对象的指针的 multiset。试编写一个有如下要求的 C++ 程序：

(1) 在 multiset 中使用函数对象 compareStudents ()，使之对象可以按学生姓名排序；

(2) 在 C++ 中，要先定义 6 个 student 对象，然后把它们插入到 multiset 中；

(3) 显示在 multiset 中元素的信息；

(4) 创建 student 对象时要有重名检查 multiset；

(5) 可以用同样的键存储多个对象。

答：

```
//multiset 自动地排序用指针存储的 student 对象
#include < iostream >
#include < set >
#include < string >
using namespace std;
class student {  //定义学生类
  private:
    string lastName;
    string firstName;
    long phoneNumber;
  public:
    student():                                    //缺省构造函数
      lastName("blank"),firstName("blank"),phoneNumber(0L){}
    student(string lana,string fina,long pho):    //三个参数构造函数
      lastName(lana),firstName(fina),phoneNumber(pho){}
    friend bool operator < (const student&,const student&);
    void display()const{  //显示 student 的数据
      cout << endl << lastName << ",ht" << firstName
        << "hthtphone:" << phoneNumber;
```

```
    }
    long get_phone()const{                          //返回电话号码
        return phoneNumber;
    }
};
//对 student 类重载 <
bool operator<(const student& s1,const student& s2){
    if(s1.lastName==s2.lastName)
      return(s1.firstName<s2.firstName)? true:false;
    return(s1.lastName<s2.lastName)? true:false;
}
//利用指来比较学生的函数对象
class compareStudents{
    public:
    bool operator()(const student *ptrS1,const student *ptrS2)const{
      return *ptrS1<*ptrS2;
    }
};
int main(){                             //指向学生 ptr 的 multiset
multiset<student*,compareStudent>setPtrsStus;
multiset<student*,compareStudent>::iterater iter;
//6 个 student 对象初始值
student* ptrS1=new student("Yankai","Wan",68389154);
student* ptrS2=new student("Liqin","Qiao",51859004);
student* ptrS3=new student("Baolin","Li",32081221);
student* ptrS4=new student("KuangLi","Wang",68386666);
student* ptrS5=new student("Baingli","han",68814321);
student* ptrS6=new student("Huiphug","Rao",42115366);
setPtrsStus.insert(ptrS1);          //把 6 个 student 对象放入 multiset 中
setPtrsStus.insert(ptrS2);
setPtrsStus.insert(ptrS3);
setPtrsStus.insert(ptrS4);
setPtrsStus.insert(ptrS5);
setPtrsStus.insert(ptrS6);
//显示 multiset 中的元素
cout<<"\n\n Set sorted when created:";
for(iter=setPtrsStus.begin();iter!=setPtrsStus.end();iter++)
(**iter).display();
iter=setPtrsStus.begin();
```

```
//删除所有 student
while(iter!=setPtrsStus.end()){
    delete *iter;                    //删除 student
    setPtrsStus.erase(iter++);       //删除指针
}
cout<<endl;
return 0;
}
```

10.6 **采用 vector 顺序包容器编写一个 C++ 程序显示排序后的结果，其要求如下：**

(1) 用户先输入单词数组，然后把排序 sort() 函数应用到该数组中；

(2) 用成员函数 push_back() 插入某一个元素，用 [] 运算符和成员函数 size() 显示元素。

答：

```
//采用串对象,push_back()成员函数、[]运算符以及 size()成员函数的顺序包容器
  //vector
#include<iostream>
#include<string>
#include<vector>
#include<algorithm>
using namespace std;
int main(){
    vector<string>vectStrings;
    string word;
    char ch;
    do {
      cout<<"Please enter a word:";
      cin >>word;
      vectStrings.push_back(word);
      cout<<"Please enter another('y'or'n'):";
      cin>>ch;
    }while(ch=='y');
  sort(vectStrings.begin(),vectStrings.end());
  for(int k=0;k<vecStrings.size();k++)
    cout<<vectString[k]<<endl;
  return 0;
}
```

第 11 章　例外处理和命名空间

11.1　填空题

⑴ 在 C++ 中，例外的关键字是_____、_____、和_____。

⑵ 命名空间支持_____和_____机制。

⑶ 用 using 语句来使用命名空间有_____和_____形式。

⑷ 例外是由_____和_____原因产生的。

⑸ 例外的 5 种典型例子是_____、_____、_____、_____和_____。

答：

⑴ try，catch，throw　⑵ using，namespace

⑶ using namespace name，using namespace::element

⑷ 程序中存在非法操作（隐式例外），程序中有意安排（显式例外）

⑸ 内存耗尽，数组下标越界，算术溢出，除数为零，无效函数参数

11.2　选择题（至少选一个，可以多选）

⑴ 当例外抛出时，_____。

a. 从抛出例外的语句开始到 try 块结束；

b. 从 catch 块开始到 try 块结束；

c. 从发现的错误点开始到 catch 块结束；

d. 从抛出例外的语句开始到 catch 块结束。

⑵ 下面可以抛出例外的错误是_____。

a. 无效函数参数；

b. 用户按下 CTRL + C 来终止程序；

c. 掉电；

d. new 不能获得所需的内存。

⑶ 例外抛出时要传递的附加信息放在_____。

a. 关键字 throw 语句中；

b. 捕获的 catch 块中；

c. 导致错误的函数中；

d. 例外类的对象中。

⑷ 匿名命名空间是_____。

a. 只有一个名字的空间；

b. 只有一个匿名的空间；

c. 命名空间是没有名字的；

d. 命名空间的名字不在此处标明。

答：⑴ d　⑵ a，d　⑶ d　⑷ c

11.3　试编写有 try 和 catch 语句的例外处理程序。倘若在程序中用 new 分配内存时，如操作

不成功，则用 try 语句抛出一个字符型例外，用 catch 语句来捕获此例外。

答：
```
#include <iostream>
using namespace std;
void main(){
  char *buffer;
  try{
    buffer=new char[256];
    if(buffer==0)
      throw  "内存分配失败!";
  }
  catch(char *str){
    cout<<"有例外产生:"<<str<<endl;
  }
}
```

11.4 试定义一个例外类 Exception。这个类用成员函数 Displayreason() 来显示例外类型。定义函数 func() 抛出例外，在 main() 函数的 try 块中调用 func()，在 catch 块中捕获例外。试编写例外处理程序。

答：
```
#include <iostream>
using namespace std;
class Exception{
  public:
    Exception(){}
    ~Exception(){}
    const char *Displayreason()const{
      return  "Exception 类中的例外。";
    }
};
void func(){
    cout<<"在子函数中抛出 Exception 类例外"<<endl;
    throw Exception();
}
void main(){
    cout<<"进入主函数"<<endl;
  try{
    cout<<"在 try 块中调用子函数"<<endl;
    func();
  }
  catch(Exception e){
    cout<<"在 catch 块中捕获到 Exception 类型的例外:";
```

```
    cout<<e.Displayreason()<<endl;
  }
  catch(char *str){
    cout<<"捕获到其他类型的例外:"<<str<<endl;
  }
  cout<<"返回到主函数,例外已被处理!"<<endl;
}
```

程序输出为：

进入主函数
在 try **块中调用子函数**
在子函数中抛出 Exception **类例外**
在 catch **块中捕获到** Exception **类型的例外：**Exception **类中例外**
返回到主函数，例外已被处理

第 12 章　面向对象建模

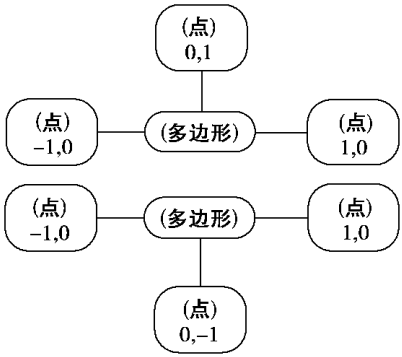
12. 1　**试按 OMT 方法做一对象图。要求按下述条件画出具有公共边的两个三角形的实例图：**

⑴ 只属于一个多边形的一个点；

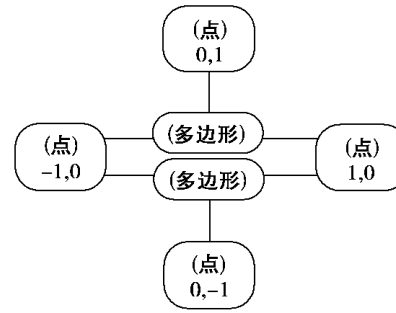
⑵ 属于一个或多个多边形的一个点。

答：⑴ 如图解 12-1 中所示，一个实例图具有公共边的两个三角形，其中公共边的每个点准确地属于一个多边形。

⑵ 如图解 12-2 所示，一个实例图具有每个点可以共享的公共边的两个三角形。

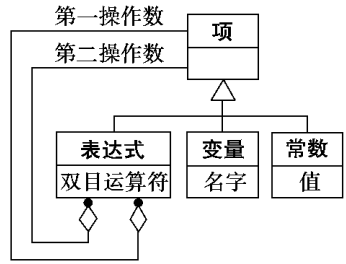


图解 12-1　每个点准确地属于一个多边形的实例图



图解 12-2　每个点能够属于多个多边形的实例图

12. 2　**给出表达式 (x + y/2) / (x/3 + y) 的类图 (见图题 12-1)。**



图题 12-1　简单算术表达式的类图

试问：

⑴ 做出该类图和实例图。在表达式中括号用于分组，但该图中不需要。重数表示项可用于多个表达式之中；

⑵ 修改该类图，使之项不被共享并能处理单目减。

答：⑴ 表达式 (x + y/2) / (x/3 + y) 的实例图在图解 12-3 中示出。在练习中的对象图实际上

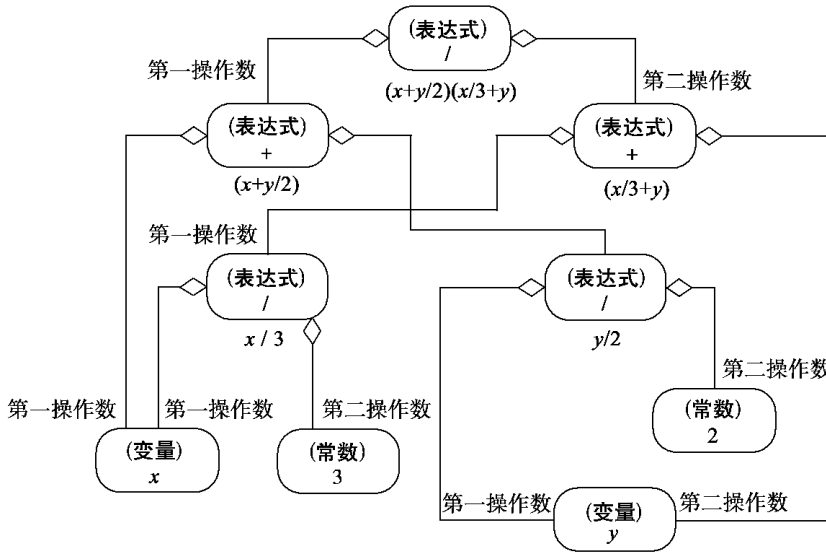
是二元表达式的一个元模型。中缀表示法要求有括号，但在元模型中是不需要的。也存在其他的表示，诸如后缀，也不需要括号。例如，在后缀表示法中，同一个表达式可变成 xy2 / + x3 / y + 1。

在这个练习的图中包含了递归。表达式可以用由表达式组成的项来构成。非常复杂的表达式可以表示成复杂的实例图。

图题 12-1 中表示的项可以被表达式共享。这种情况模拟了在习题 12. 1 中所讨论点的共享。如果在相应的实例图中重视链接的方向，则实例图是个有向图。

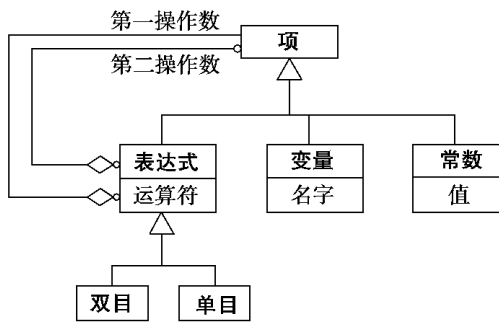
在图解 12-3 所示的实例图中，项 (term) 是作为抽象类来处理的，因此仅所示的类表达式、类变量和类常数的实例才有方向性。

在实例图中部分表达式表示得很清晰。在读取实例图中，记住角色名在关联末端项 (term) 处。



图解 12-3　表达式 (x + y/2) / (x/3 + y) 的实例图

(2) 在图解 12-4 中所示的是扩展不共享的项和处理单目减。注意因为单目运算符的可



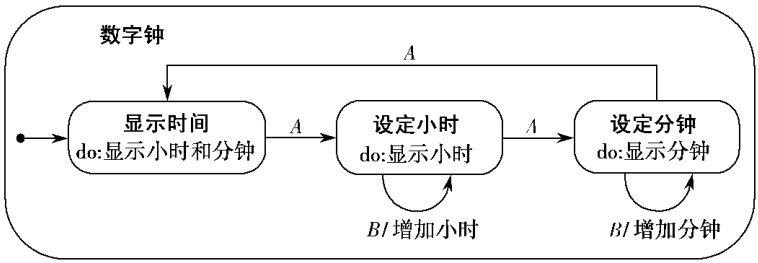
图解 12-4　简单算术表达式扩展的类图

然性，第二操作数的重数是 0-1。该图对每个项必须准确地属于一个表达式不能表达得很好。这个图可以进一步地改进，即用单一的具有资格符操作数的资格关联来替代项和表达式之间的两个关联（来改进）。这将得到在关联末端表达式上准确 1 的重数，并且可能还有其他几种变化。

12.3 一只简易的数字钟，它有显示和两个设置按钮：A 按钮和 B 按钮。钟有两种操作模式：显示时间和设置时间。在显示时间模式中，显示小时和分钟，并且用闪烁的比号（:）分隔。在设置时间模式中，有两个子模式：设置小时和设置分钟。A 按钮用于选择模式，每按一次，模式依次变化，即显示 - > 设置小时 - > 设置分钟 - > 显示等等。在子模式中，每按一次 B 按钮，小时或分钟依次递增。在设置其他事件之前，必须释放按钮。

请设计该数字钟的状态图。

答：简易数字钟的状态图，如图解 12-5 所示。我们假设按压下一个按钮是一个事件并且我们可以忽略一个按钮的释放。用状态图中 A 表示按压下 A 按钮，同样用 B 表示压下 B 按钮。



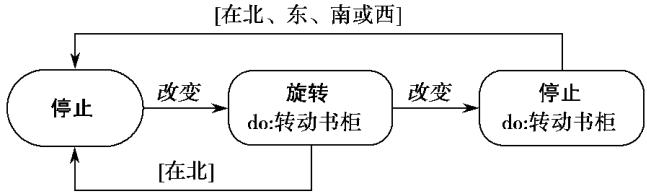
图解 12-5 简易数字钟的状态图

12.4 在探索一个古老城堡的时候，你和你的一位朋友发现了一个秘密通道入口的书柜。在检查该书柜时，你的朋友从烛台上拿走蜡烛并发现这个烛台是个入口的控制装置。这时，书柜转了半圈，把你推进了秘密通道。你的朋友将烛台复原，这次书柜转了一圈，仍然把你关在书柜后面。你的朋友拿出蜡烛，书柜又开始了转动，但你连忙用身体挡住，你的朋友把蜡烛递给你，你们一起将书柜保持在半开的状态，但这使你的朋友留在了后面而你在前面，你放回蜡烛。当书柜开始转动时，你拿出蜡烛，书柜转了四分之一圈后停止。你和你的朋友也就可以继续探险了。

开发与上述脚本一致的书柜控制状态图。为了尽快进入通道，应该先做些什么呢？

答：这个练习演示了即使一个简单的状态图也能够导致复杂的行为。在这个练习中给定状态图（如图解 12-6 所示）能用脚本加以解释。“改变”事件产生在每当蜡烛从烛台拿走或每当蜡烛放回烛台时。每当书柜在墙后面时，满足“在北”条件；每当书柜在前面、后面或一边时，满足“在北、东、南或西”条件。

当你首先发现书柜时，它位于停止状态点南面。当你的朋友移动蜡烛时，“改变”事件使书柜进入旋转状态。当书柜指向北时，“在北”条件使书柜放回到“停止”状态。当你的朋友重新插上蜡烛时，另一个“改变”事件将书柜放回到“旋转”状态直至再一次指向北为止。把蜡烛拿出产生另一种“改变”事件，并且如果你没用你的身体堵塞住的



图解 12-6　进入秘密通道控制的状态图

话，即可产生书柜整个的旋转。外力使书柜放回原处超出了控制范围并且在此也不作解释。

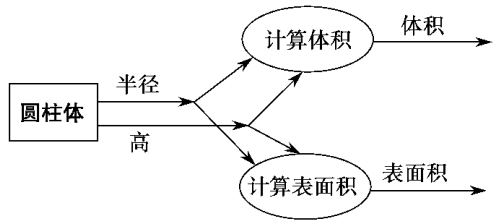
当你再一次把蜡烛放回到另一个“改变”事件时，使书柜再一次进入“旋转”状态。把蜡烛放回导致产生另一个“改变”状态的结果，使书柜进入了停止状态。当书柜转了 1/4 圈后，条件“在北、东、南或西”被满足，书柜进入停止状态。

如你要立即获得进入秘密通道的话，必须首先在书柜完成旋转 1/4 圈之前，拿走蜡烛并迅速放回才能实现。

12.5　给出计算圆柱体表面积、体积的数据流图。输入是圆柱体的高和半径，输出是体积和表面积。试讨论这个数据流图的几种实现方法。

答：

图解 12-7 包含了计算圆柱体几何的数据流图。



图解 12-7　计算圆柱体体积和表面积的数据流图

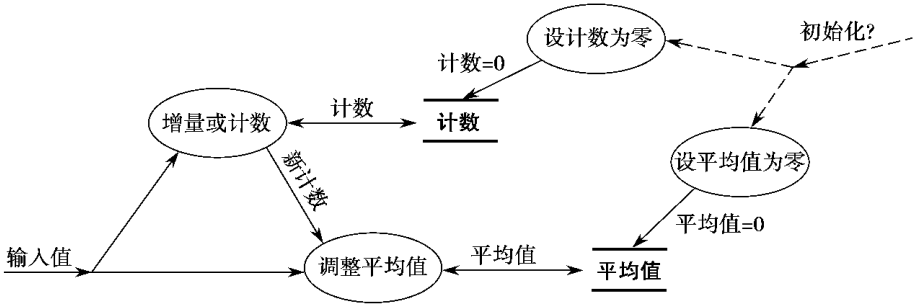
对圆柱体的计算体积和表面积有一些其他不同的方法，它们是：

- 用公式计算。体积 = $\pi r^2 h$，表面积 = $2\pi rh$，其中 r = 半径，h = 高。
- 用查表方式计算。
  体积和表面积被列在标准的半径和高的值的表上。
- 用数值方法计算。

用差分方程中计算圆柱体的体积和表面积的方法。

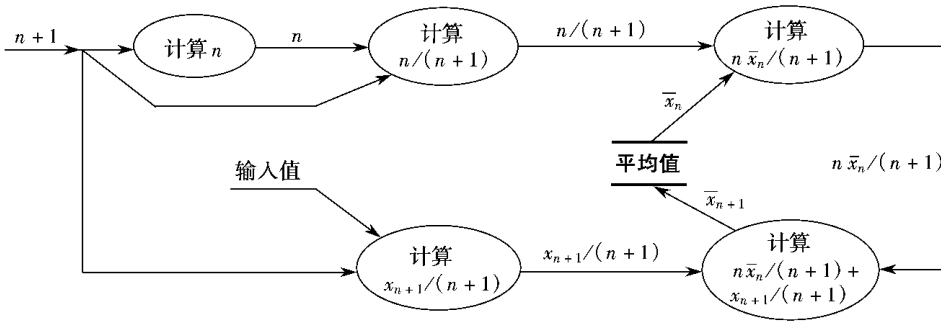
12.6　使用二次方程为起点画出流程图，求二次方程 $ax^2 + bx + c = 0$ 的根。实数 a，b 和 c 是输入，输出是满足方程的 x = R1 和 x = R2 的值。记住，R1，R2 可以是实数或虚数，依赖于 a，b，c 的值。二次方程的 R1，R2 为（- b ± SQRT（b^2 - 4ac））/（2a）。

答：图解 12-8 表示了计算连续输入值的平均值的数据流图。



调节平均值过程

(注意：$n+1$=新计数，x_n=第n个输入值，$\bar{x}_n$ = n 值后的平均)



图解 12-8　**计算连续输入值的平均值的数据流图**

第 13 章　面向对象设计与实现

13.1　许多系统中的普遍存在的问题是掉电或硬件出错时如何保存数据。理想的解决方法应对系统来说是可行的、低成本、小型、快速、维护容易。当然也应该对热、灰尘、湿度等具有抗干扰性。各种可行技术之间折衷往往影响到功能需求。

根据上述想法试比较下面的每种解决方法：

⑴ 每次系统加电时复位所有数据。

⑵ 从不关掉电源。如有必要使用特殊电源供电，包括后备电源。

⑶ 把关键信息保存在磁盘上。定期复制所有或部分数据到磁盘上。

⑷ 当系统关闭后，用电池为系统存储器供电，用此方法能持续提供有限的电能。

⑸ 用一个特殊的存储部件，诸如磁泡存储器或可电擦可编程的只读存储器。

⑹ 通过开关由用户输入关键参数。这种用途的商品化开关有好几种，如集成电路的集成触发开关。

答：

读者应当注意这种问题不会给出详尽无遗的解决方法。下面只对疑难的⑴、⑶、⑸给出解答。

⑴ 每次系统加电时，复位所有数据，不用担心。

这是最廉价最简单的方法。是对程序相对地容易的作法，因为在加电时都需要初始化所有的例行程序，以便允许用户键入参数。但是，这个方法不能对系统提供必须地连续服务，或者在断电期间不一定丢失数据。

⑶ 把关键信息保存在磁盘上。定期复制所有或部分数据到磁盘上。

这个方法减少了花费和沉重的负担。在电源断电事件发生之后，系统停止运转。要求操作系统妥善处理磁盘和磁带驱动。要求操作管理磁带，以排除一切无联系操作的应用程序。

⑸ 用一个特殊的存储部件，诸如磁泡存储器或可电擦可编程的只读存储器。

这个方法相对廉价和自动化。但是，当断电时，系统不能运转。某些限制可能发生，诸如限制保存数据次数或者限制保存大量数据。一个程序可以要求在断电时保存重要的参数。

13.2　在上一习题中，对下面的系统选择一种或多种数据存储策略。对每种情况给出估计需要多少字节的存储空间的理由。

⑴ 四则运算袖珍计算器。主要电源是太阳能，执行基本的算术运算。

⑵ 电子打字机。主要电源是可充电电池或交流电源。以两种方式操作。一种模式是一次打文本的一行。在打印前对每行进行编辑，液晶显示 16 个编辑字符。另一种模式是在打印以前输入和编辑整个文本。希望保存工作文本在一年以上直到电源用尽。

⑶ 个人计算机的系统时钟。主要电源是个人计算机提供的直流电。给计算机提供时间

和数据信息。必须保存正确的数据 5 年时间，直到电源用尽为止。

⑷ 航班预订系统。主要电源是交流电，用于预订航班座位。系统必须在任何时间和任何费用下都保持运行。如果由于某种原因，系统必须关闭的话，任何数据不能丢失。

⑸ 马达的数字控制和恒温保护单元。根据测量到的电流和马达散热，计算出马达的温度，对于大功率设备提供了恒温保护。如果计算的马达温度超过了安全值，必须关掉马达，并且在冷却以前不允许重新启动。电源是交流电，它可以被中断，一旦电源打开，系统必须提供保护。用于恒温模拟的参数最初是由工厂设置，如果必要的话，在系统安装后，可以修改。因为马达温度不能直接测量，所以有必要在关掉电源后至少一小时继续模拟马达温度，以防止电源在马达冷却前重新恢复。

答：

⑴ 四则运算袖珍计算器。

不用担心所有永久性数据存储。考虑其他所有的选择太昂贵。这种计算器售价几块钱，典型应用于收支平衡差额存折，内存要求大约 10 个字节。

⑶ 个人计算机的系统时钟。

仅要求几个字节，但时钟必须与主电源断开后连续不断地运行。电池后备是廉价的，要求时钟电路设计确保电池可运转 5 年时间。

⑸ 马达的数字控制和恒温保护。

大约需要 10 到 100 个字节。这种应用是需要花钱的。考虑到采用不间断电源太昂贵，对该应用的恶劣环境采用磁带和磁盘驱动太容易损坏，因此使用组合开关、专用内存部件和电池后备。开关是键入参数的好办法，因为无论用什么方法都要求有接口。专用内存部件能够保存计算的数据。电池在断电之后可用于连续地操作，并且可以显示这个应用的维护问题。我们分析这种要求，寻找替代办法，诸如当第一次开动马达时，假定马达发热，或者使用马达温度测量传感器测得马达过热。

13. 3 描述包括软件开发环境在内的如何使用工具的情况，这些工具包括浏览器、编译器、解释器、符号调试器、系统生成器和改变控制系统等，用这些工具帮助解决以下问题：

⑴ 在子系统库中，使用了不熟悉的子程序，有不太多的文档，你怎么能够快速地找到工作上所用的子程序？

⑵ 在你的程序中似乎有一个毫无意义的错误，它是存在于源代码的某一个程序模块之中。现在不能保证你所看到的刚生成的源代码能够运行。在运行连接之前，忘了重新编译这个模块是很可能的。

⑶ 你所开发的程序因内存缺欠和错误退出了。在失败的那一行，每次运行这个程序时其结果都是不同的，这依赖于每次赋予的数据。这个问题可以重复相同的数据。

⑷ 你是一个大型软件项目上的团队领导，应确保一次仅一个人工作在一个文件上。所有模块的最新版本必须作为整体来访问，但单个用户按整体调试目标来调试它自己的私有拷贝。

⑸ 每次对你所有执行中的改变，都存在着大量繁多的步骤来建立应用程序，并且这些所有步骤使你感到很麻烦，因此你试图单独地解一个需要重做的应用程序，作为你

最新版本的结果。

⑹ 你已经编写了自己的预处理程序，所使用的调试器显示了预处理程序的输出，而且你也有跟踪错误并返回原始源代码的麻烦。

答：

⑴ 解释器提供了一种方便的方法，使在库行为、旁路编辑、编译、链接和执行周期中遇到的编译语言中，快速查找到所用的子程序。某些语言同时兼有编译器和解释器。解释器习惯用于快速程序开发，而编译器习惯用于产生程序高效的最后代码版本。

⑵ 推荐系统建立时避免这种类型的错误。在你修改一个或多个源文件之后，它保证所有要求的步骤和仅要求重建应用程序都能被执行。换句话说，你面对重新建立任何事情的抉择，其中时间是要花费的，或试图记住影响你改变易于产生错误的步骤。

⑶ 符号调试是诊断这种类型问题的方便的工具。它允许运行你的程序直到错误条件发生为止，然后看到发生错误的那一行以及程序变量的值。它是出现在程序运行到内存之外。

⑷ 改变控制系统是调整团队软件项目的极好方法。

⑸ 系统生成器能解决这个问题。

⑹ 某些语言提供这个问题。例如，在 C 语言中存一种行号构造，预处理器能够插入到输出指向另一个文件一行的那个点上。

13. 4 描述处理下述内存管理问题的策略，假定自动垃圾收集是不能用的。你的回答应当采用在编码期间程序员可以使用的准则。

⑴ 文本操作系统。一个公共的操作是从几个较小段的数组中用连续的内存创建一个大的文本段。该系统希望处理很多文本，并且你不能浪费内存，你既不能设定超出文本长度的范围，也不能够超出数组的长度。如果这些模块作为库子程序是不能用的话，下面的操作是你必须提供的生成模块，决定文本段的大小，分配给定大小的内存段，释放以前分配的内存段，把文本从一个内存段拷贝到另一个内存段并且组合两个文本段其结果放在以前分配的内存段。编写组合文本的方法伪码，使不再使用的内存空间恢复原状。

⑵ 多遍扫描编译器。动态地创建对象，每遍扫描检查上一遍扫描创建对象并产生下一遍使用的对象。有编译器的计算机系统将运行有一个实际上无限的虚拟空间和具有交换算法的操作系统。动态地分配和释放内存的运行库的子程序效率不高。讨论两个选择方案的相对优点：

- 舍去所有的垃圾收集机制，让操作系统分配一个大量的虚拟内存；
- 当对象不再引用时，小心释放内存。

⑶ 银行软件或运行长时间的空中运输控制系统。你有像 13. 4 ⑵ 练习题中所描述的相同的计算机系统和运行库。讨论两种方法的相对长处。

⑷ 使用大块内存来创建并返回一个对象的子程序。在对象类上仅有的操作是，用你打算编写的其他子程序来实现的，而不希望在周围留下垃圾。讨论下面两个方法的相对优点：

- 每次调用子程序就删除最后一次创建的对象，如果有的话；

- 每次调用子程序，它可以创建一个新对象。当不再需要你编写的调用子程序时，它依赖于删除该对象所调用的子程序。考虑两种情况：一是子程序调用是递归的；一是子程序调用不递归。

答：

⑴ 因为紧凑的内存要求文本段不再需要时释放掉。这个问题是当文本段不再需要时确定一致性的开放策略。类和方法提供了一个解的方便框架。准则之一是用修改文本段的方法完成对内存回收。对文本段所拥有的对象外面的访问应当是只读的。暂存的文本段应尽可能快地释放。

文本段组合的一种方法是：

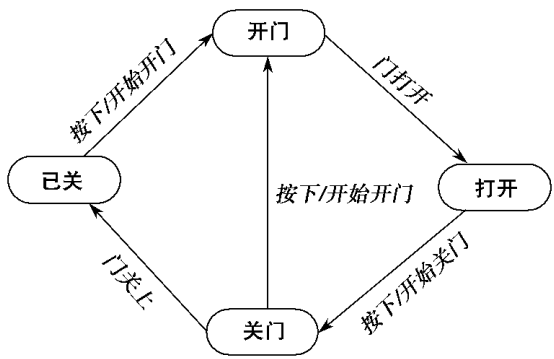
```
Determine the size of the segments that are to be combined.
Allocate enough memory for the result.
Copy the original text segments into the allocated memory.
Dellocate the memory assigned to the original text segments.
```

⑵ 对多遍扫描编译器，它对简单地让操作系统分配大量的虚拟内存来说是较好的策略。需要内存的数量依赖于被编译的源代码的大小。因为程序被分割成容量适度的模块，所以它可能放在内存合理地所需要的数量范围之内。

⑶ 你不能让操作系统分配大量的虚拟内存并忽略有关在不确定运行系统中的垃圾收集，最后所有内存将被消耗殆尽。当对象不再引用时，内存必须释放。

⑷ 第一个方法，释放在第一个地方中分配的子程序的内存。当内存块不再需要之时，保证内存回收而无需许多其他子程序完成对内存回收工作。但是这样做递归过程是很复杂的。

13.5 图题 13-1 是车库门开动时的状态图，是用程序中的位置状态来实现的。试用伪码或任何其他结构化程序设计语言模拟车库门开动的状态。



图题 13-1 车库门开动状态图

答：

对车库门开动的伪码列在下面：

```
<closed>wait for depress event
<opening>start opening door
```

```
            wait for door open event
<open>      wait for depress event
<closing>   start closing door
            wait for either depress or door closed event
              if depress event then goto opening
              if door closed event then goto closed
```

不用担心使用了 goto 语句！它们在表示控制流上（诸如例外和中断）使用有一定的合法性。

读者也可仿此用结构化程序设计语言重写一遍。

第 14 章　C ++ 面向对象设计与实现的典型实例剖析

14. 1　试用面向对象方法 (OMT) 的三种模型 (对象、动态和功能) 分析并实现电力分布式设计系统的单线图编辑器 (OLIE——One-LIne diagram Edifor) 的 CAD 工具。要求用母线或电路方式连接网络部件来布局电力分布式系统。OLIE 编辑器可以用固定菜单、弹出式菜单和有向图直接输入等多种方式加以控制。

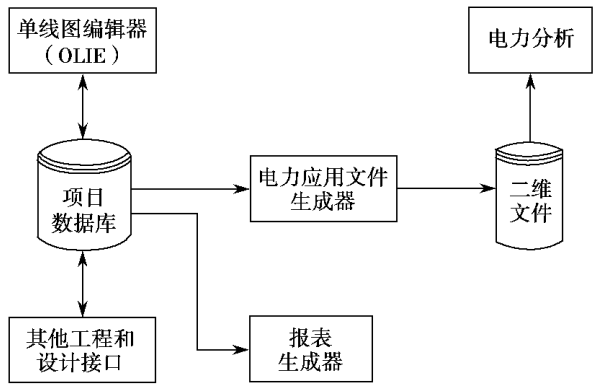
答：计算机辅助设计 (CAD) 工具是一个软件系统，它能促使所有设计活动或部分设计活动自动化。CAD 工具，具有代表性的有交互作用的用户界面和内涵的图形。当前 CAD 工具应用的领域有绘图、化学反应过程设计、机械设计和分析、HVAC (供热、通风和空调) 集成电路芯片设计、印制板电路布局和管道测试设备设计。

单线图编辑器 (OLIE) 是设计电力分布式系统的 CAD 工具，该系统允许工程师用母线或电路方式连接网络部件来布局电力分布式系统。这种编辑器采用固定菜单、弹出式菜单和有向图直接输入等多种方式加以控制。

跟大多数用户界面的应用一样，OLIE 主要以动态模型为主，对象模型也很重要，但功能模型相对就不太重要了。开发 OLIE 最困难的地方是要求与几个现有的子系统集成，这些现有子系统是图形系统、数据库管理器和菜单系统。这些子系统有不同的特点、缺陷和性能上的开销。为了提供真实自然的用户交互作用，OLIE 必须克服这些问题。

- **问题陈述**

决定功能需求有几个因素，包括市场调研、顾客需求、其他图形编辑器的审查以及电力工程师设计 OLIE 软件的经验。市场调研预测表明，OLIE 与其他用于体系结构工程的软件集成，并通过公共工程项目数据库构成一个合理应用，是用户所期待的，如图解 14-1 所示。



图解 14-1　OLIE 与其他应用的集成

1. 功能需求

OLIE 必须用数据库管理系统 (DBMS) 把数据存储在数据库中。首先，这似乎可以把不合适的“需求”和出现的可能的决策，能够从“真实”的需求中分析出结果来。但是，OLIE 软件不期望在它本身孤立地完成，至少电力工程师完成其工作时，需要有一个合适的工具。这样，OLIE 必须对其他电力工程师提供接口。对这种大型系统的需求分析最终导致 OLIE 必须使用数据库的结论。

根据定义，OLIE 必须提供编辑单线图的基本命令，诸如创建、扫视、缩放、拖曳、移动、拷贝、挑选和存放。OLIE 必须有使用户的思维链不中断的真实的响应，用户接口必须对工程师而不是对计算机专家适用。

OLIE 必须允许数据在用户之间、应用程序和项目之间共享。该系统还必须允许在扩充工作期间锁定设计的一部分，这些扩充持续锁定工作时间，可能几天或几个星期，与通常的数据库完成一件事务仅需几秒钟正好相反。典型的 OLIE 用户一次有几个激活的项目，那么多个工程师可以在各自的项目上工作，信息库 (诸如绘图符号) 可以跨越许多项目。

OLIE 必须弥补存储媒体的错误、计算机故障和软件错误。软件必须能够移植到其他硬件、操作系统和软件子系统上，包括新的图形硬件。OLIE 必须扩充包括新功能新应用在内的程序，并且要容易维护。工程师必须能够用几种不同格式生成有关电力分布式系统的报表。

2. 体系结构要求

管理要求 OLIE 建立在几个内置的软件包上：

⑴ 创建、选择和编辑符号、正文和直线的图形系统。

⑵ 插入、删除、更新和查询数据库的接口。

⑶ 建立显示表单和菜单，并辨认用户输入的一种用户接口系统。

⑷ 内置的面向对象程序设计语言。

- 分析

1. 对象模型

对象模型包括五个模块：打包 (封装) 模块、电力特性模块、几何图形模块、分组模块和连接模块。打包模块包含了项目的高层结构，一个项目由多个单线图绘制而成，其中每个单线图进一步由多个页组成。电力特性模块定义了每个电力部件的属性。几何图形模块描述屏幕上呈现的电力项目。连接模块获取单线图和电力网络之间的结果。

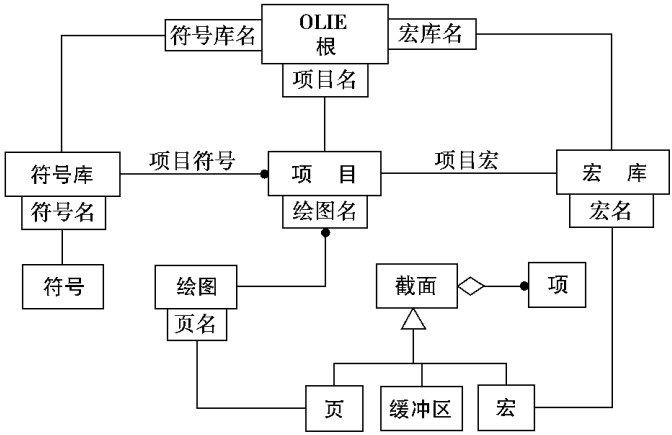
图中没有显示实际的模型中更详细的细节，这里只显示了基本结构。我们已经屏蔽了模块之间共享的类。

⑴ 打包 (封装) 模块

打包模块 (图解 14-2) 用分层方式组织用户访问数据。类 OLIE 根有单一实例，提供库和项目名的上下文。顾客可以在同一时间上激活一个或多个项目，用各自的项目名标识。在每个项目内有几个绘图，每个均用绘图名来限定，每个绘图名在项目中是惟一的，但在其他项目中可以重用，一个绘图也可有多个命名的页。

可以有几个符号库和宏库。每个项目指定一个符号库和一个宏库。符号是一个图标，在单线图的上下文中是有意义的。例如平行的锯齿线表示一个变压器。宏是用户定义的符号、

母线和电路的分组。一个宏或一个符号用它所在库内的名字来限定，所以同名可以在不同的库中使用，一个库可以用于几个项目之中。

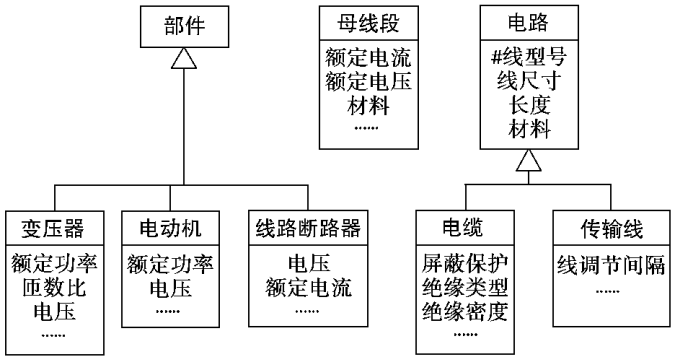


图解 14-2　打包模块

截面是单线图的矩形区域，它包含项的聚集，诸如符号、母线和电路。一个截面是对图进行访问控制的最小单位。因为不同的截面可以由不同的工程师并行地编辑，所以页、缓冲区和宏都是截面。一个页是包含部分单线图项目层次中的最低层，缓冲区包含引用部分单线图的编辑操作。页、缓冲区以及组成包含所有相同类型的项，可以把它们概括成一个截面，这实际上允许了代码的可重用。

⑵ 电力特性模块

电力特性模块（图解 14-3）定义了电力分布系统中部件的电力特性。这个模块的目的是定义类的属性，这个模块不包含关联。电路纳入两个项：电缆（地下）和传输线（地上）。每个从电路中继承公共属性。母线段有一个简单结构。部件归纳为许多不同的种类，诸如变压器和线路断路器。在这个模块中的属性不出现在单线图上，这些数值由工程师在弹出菜单上键入。

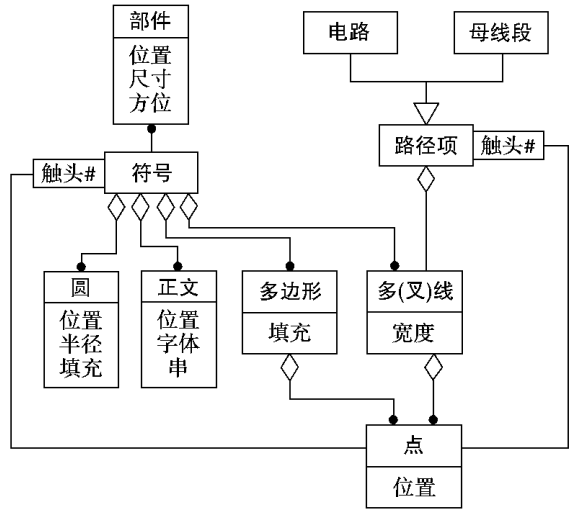


图解 14-3　电力特性模块

⑶ 几何图形模块

几何图形模块（图解 14-4）描述了单线图在屏幕上的布局。电路和母线段（总称为路径

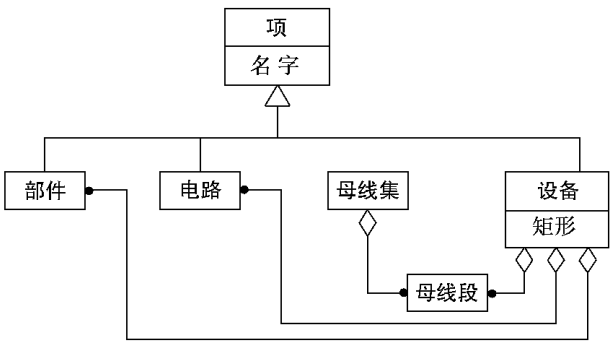
项) 由用户设定宽度，用多 (叉) 线绘制。部件从符号库中挑选任一符号表示，每个符号是用几何形状构造。各个符号有一个预先连接点或触头的固定号码，可以连接图中其他符号的触头。每个路径项有两个触头，每个多 (叉) 线的末端有一个。一旦两个触头被连接，它们保留项在屏幕上移动连接。该图为维护连接需作必要地调整。



图解 14-4　几何图形模块

⑷ 分组模块

OLIE 支持项 (item) 分组的两种类型：设备和母线集 (见图解 14-5)。设备是作为单一装置的采购电力部件的分组，通常在金属柜子中，并在所示的单线图上加入少许矩形封装在相应的装置、电路和母线中。母线集是单线图中母线段的组合，由单一的物理单元组成。设备和母线集类似于各自是一个或多个项的集合，项至多可以属于一个集合。

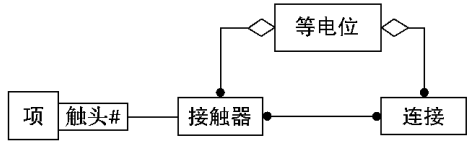


图解 14-5　分组模块

⑸ 连接模块

连接模块 (图解 14-6) 与单线图中电力系统电力连接的物理连接有关。单线图编辑器和电力分析程序各自有点稍微不同的电力系统视图。编辑器看作由符号、母线和电路相互联系的网络组成的单线图。编辑操作 (诸如移动、拷贝和剪切) 必须快速确定断开的连接，并且

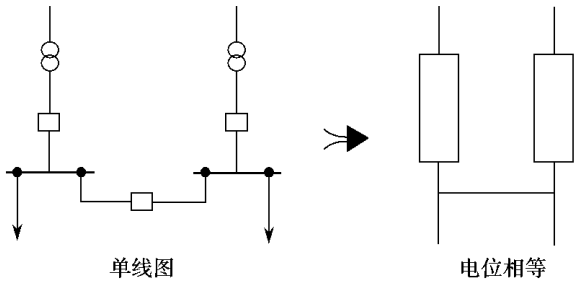
伸长或压缩母线和电路。电力分析程序作为电力电路的电力系统的一个视图：支线和等电位的表示，类似于一个边-顶点有向图。支线是两个端口电路元素，与电流通过分支线到穿过其他不同的支线电压有关。支线的端口与等电位一起连接的，连接在等电位上的所有端口有相同的电压。输入等电位的电流总和是零。



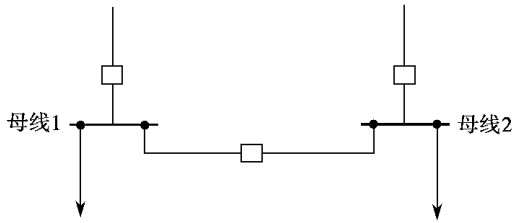
图解 14-6　连接模块

在单线图中，从符号、母线和电路映像到作为相当的电力电路的支线和等电位不是不重要的。在图解 14-7 表示出一个简单的例子，随着线路断路关闭，两个母线是等电位的。

一件复杂的事情是为支线和等电位命名。为解释分析程序的结果，必须惟一标识支线和等电位。首先想到是应用简明的符号、电路和母线来命名，但是有几个复杂的事情发生了。一个符号的电力模型取决于所表示的是什么。某些符号（诸如变压器和电路）表示电路支线，而其他的符号（诸如开关和母线）表示连接。用户可能对等电位疏忽大意地指定多个名字，而在开关任一边用母线命名，如图解 14-8 所示。



图解 14-7　单线图映像到电位相等电路



图解 14-8　对母线指定的命名意义不明确

事实上，OLIE 应该在单线图创建的任何时候命名电力模块，要检查名字的惟一性。最坏的情况，当建立电力应用的输入文件时，应有一种完美无缺的方式来保证名字惟一。

在图解 14-4 的几何图形模块中，符号、母线段和电路对构成的连接有一个触头。等电位、接触器以及连接映像到单线图的符号、母线和电路以及支线和结点的表示之间。支线在模块中没有显式表示，因为它们很容易确定。

2. 动态模型

因为 OLIE 是交互式编辑器，所以动态模型和用户接口基本上是一回事。在对象模型经过反复修正迭代，使之基本达到符合要求之后（即很少再发生修改），就开始设计用户接口。

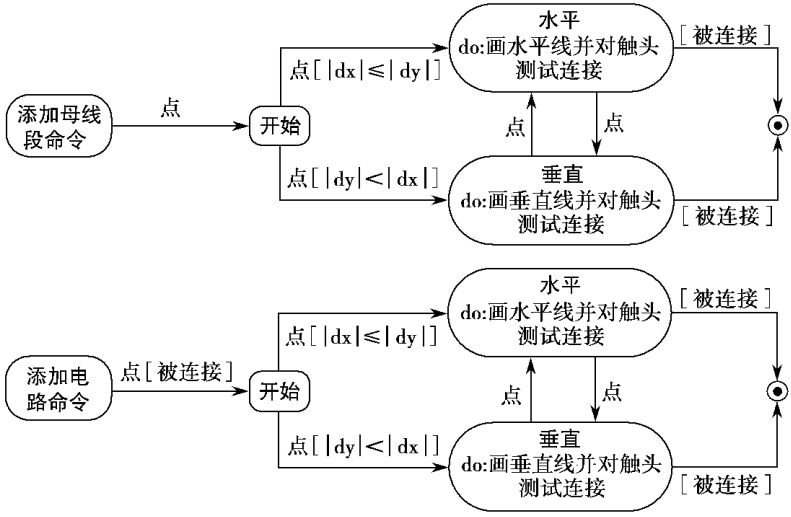
我们检验其他的图形编辑器并选取最好的特征，进而将用户的交互作用从功能上分组，诸如显示控制、编辑、遍历项目层次等等，并加上开发的详细说明。

然后，我们设计菜单并指定鼠标行为。左鼠标按钮从单线图中或固定菜单的命令中选取一些项，右按钮从弹出菜单选取命令，中间按钮用于扩充选项。有时，选择一个菜单项可能产生了另一个菜单项，我们规定菜单嵌套的深度为3，因为再深的嵌套对用户容易造成困惑。

我们对原子操作，诸如选择、移动、拷贝、删除、镜像和显示正文，挑选说明。对每个操作构造动态模型，其中一些操作讨论如下。

⑴ 添加母线段和电路

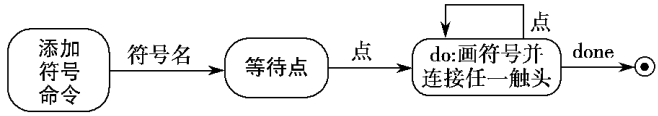
用户建立一系列交替选择的水平和垂直线段的鼠标操作。OLIE 自动地连接符号和其他电路或母线。图解 14-9 表示了添加母线段和电路的动态模型，这两个动态模型稍微有点不同，因为母线段可以“悬挂”，无需连接任何部件，而每个电路是在一对触头之间运作。



图解 14-9　添加母线段和电路

⑵ 添加项

项可以是符号或宏，它们用库中的名字来标识，并用在单线图（图解 14-10 所示）所画的项的图像位置确定。符号的多个拷贝可以在一系列点上用点击（click）方式添加，用户必须指定用选择 done 来停止添加项。

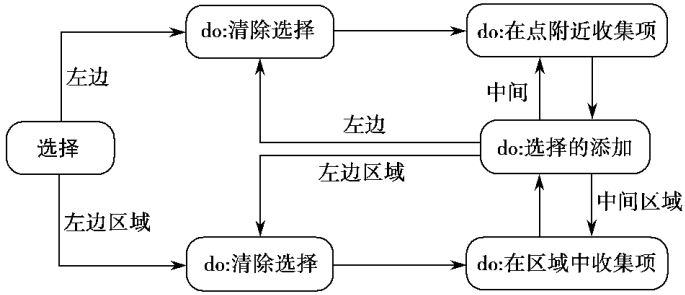


图解 14-10　添加符号

⑶ 选择几何图形

图解 14-11 中，母线、电路、符号和正文都是独立的选取，然后进一步地进行操作分组。在鼠标按钮开始一个新的选择，而中间按钮对现存的选择添加一个项。如果任一按钮按压下和释放，那么项（如果任意）靠近鼠标的点就选上。如果任一按钮按压下、保持、滑动

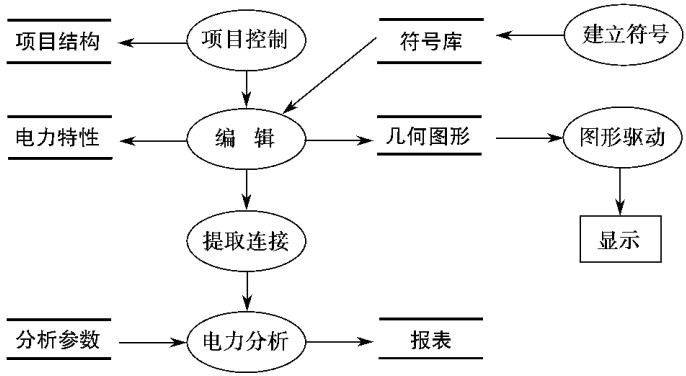
和释放的话，则选择点上或点下之间包围的矩形的所有项（如果任意）。压下、保持、滑动和释放的细节未显示在动态模型上。这样的细节属于获取鼠标事件的图形驱动模块。相反，图中展示的是左边和左边区域的具体逻辑事件。



图解 14-11　选择项

3. 功能模型

OLIE 的功能模型用图解 14-12 表示。因为具有许多交互作用的编辑器，所以系统中功能大多数是不重要的。系统的目的是与用户建立交互的数据结构，对编辑器或项目控制来说，它不是用于低层的数据流图。电力分析仅对部分功能模型有意义。电力分析用外部独立应用（不受操作系统控制）的程序执行，所以内部功能模型一般是不能访问的（但可用于每个分析的输入列表）。提取连接处理仅是分析的一部分，也是 OLIE 系统本身的一部分。

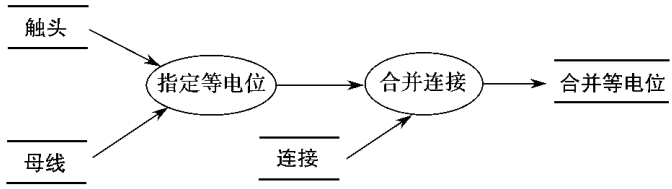


图解 14-12　OLIE 系统的功能模型

• 连接提取

应用程序准备输入文件最多的工作是图解 14-13 所示，将符号、母线、电路、触头和连接转换到支线和等电位中获得。转换算法开始对每个触头指定一个等电位，所有给定的母线的触头指定相同的等电位，并给定与母线相同的名字。符号和电路触头给定它们自己的无名等电位。下一步是扫描所有连接，合并每个进入一个等电位的等电位集合。给定合并等电位的名字，这取决于许多名字开始是如何合并的。如果不存在名字，合并的等电位不给定名字；如果存在一个确定的名字，则合并的等电位就给定这个名字；如果存在几个名字，则合

并的等电位给出其中一个名字，并且发出一个警告。



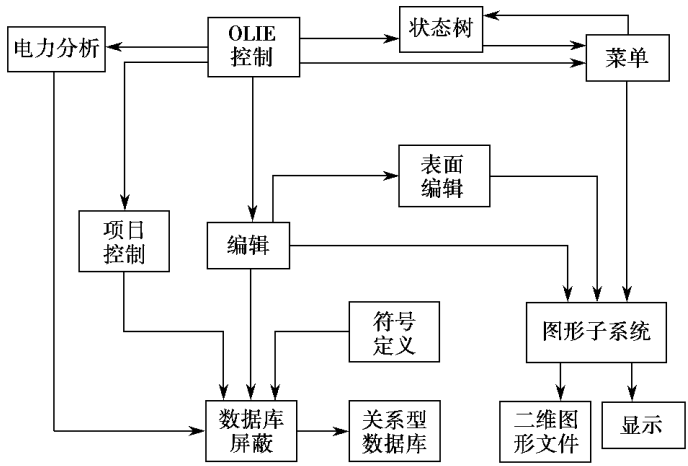
图解 14-13　**连接提取**

• 系统设计

我们感觉到不可能同时满足这样的要求：在数据库中存储单线图；支持数据共享；并提供快速真实的用户接口。我们考虑到 OLIE 直接与数据库相互作用的可能性，但测试以后发现支持真实的用户接口后响应太慢。

一个相关的问题是数据库的同步和重新运作。除数据库外，图形子系统在二维文件中存储它的数据。这就产生了如何将二维文件与数据库同步问题，并重视数据库重新运作问题。

我们考虑了几个系统体系结构。最终解（图解 14-14 所示）使用了数据库屏蔽子系统。屏蔽子系统拦截所有数据库操作，并当可能之时把这些操作转换成内存操作，在限定的区域内访问关系数据库。屏蔽子系统要求大部分数据必须在数据库之外被检出（checkout）、以内存速率操纵，然后再进入数据库检查。在这种应用中，数据库检出（或称出库）不太困难，因为不允许两个工程师同时编辑设计相同的部件。



图解 14-14　OLIE **系统的体系结构**
（箭头表示在供应者模块上对客户模块的依赖性）

操作访问通过屏蔽模块存储数据。因为关系数据库自动维护并发控制，每个 OLIE 实现可以作为单用户程序来操作，无需担心并发问题。

用图形子系统操作访问图形显示，包括单线图的显示和在二维文件中存储图形信息。编辑模块维护数据库和二维文件的一致性，在数据库中存储冗余的图形信息并提高二维文件的

性能。正常情况下，图形系统在二维文件中存储图形信息，并屏蔽系统在数据库中存储图形和其他信息。在二维文件中的信息事件变得与数据库不一致了，这时屏蔽系统应重新通过图形系统来创建二维文件。

权衡追加的内存和性能以及功能的存储空间，这个系统的用户总是要求高性能工作站，所以这种折衷是合理的。

单线图中名字必须惟一。我们决定推迟检查名字的惟一性，直到用户做完编辑为止。增量式名字检查在多用户之间限制了并发性以及用户表示满足推迟的检查，这些都是很复杂的。

数据库检出 (checkout) 也允许 OLIE 支持长事务 (long transaction) 表示。长事务，即这个事务可以持续几天或几个星期，传统的数据库与此相反，通常事务几秒钟就完成了。工程师在长事务执行期间内可以产生不一致性，这样工程师可以自由地从事他们的工作并在相同的时间内 OLIE 可以满足数据库完整性要求。

控制实现采用事件驱动方式，并基于显示状态图组织成一棵树，称为状态树。所有输入事件在树内产生状态变迁，菜单可以由状态树控制，也一定由状态树控制；依次，用户菜单选择产生输入事件由控制来解释。在树中变迁强制执行一个动作，诸如剪切、粘贴和移动。我们用状态树实现控制。

• 对象设计

对象设计是简单明了的，它给出了分析和体系结构决策。图形、数据库、状态树、菜单、表单编辑和电力分析子系统所有都要预先定义。这些子系统的主要工作是设定合适的参数。要考虑菜单布局、表单和项属性的工作，但在这里，我们不再呈现其细节。大部分工作是引用编辑器：高层交互控制流 (策略) 和低层操作 (实现)。

1. 状态树控制

因为用户接口所具有的重要性，所以我们把注意力放在动态模型上。我们采用称为状态树的状态-事件模型变化来控制用户接口 (见 12.2 节)，状态树是将状态组织成树并对隶属于树结点的事件共享公共结构和行为的一种技术，允许分层分解控制结构和消去冗余信息。事件归属于状态如何说明解释事件以及能够被子状态继承，就如同在对象类上属性和操作一样。入口和出口动作也隶属于状态说明进入和退出状态的结果。因为状态被组织成一棵树时，子状态就不能被键入 (无需键入它的超状态)，以便入口 (和出口) 动作也可以被继承。

在超状态表示的树结点内，可以嵌套子状态。在树结点上入口和出口动作等价于在嵌套状态略图内的入口和出口动作。

状态树自然而然地支持菜单驱动界面。每个菜单入口产生一个给定状态的变迁。任一菜单入口可以被任一状态选取，因为当前状态的退出动作蕴含着清除一切，而不管是什么样的用户命令。

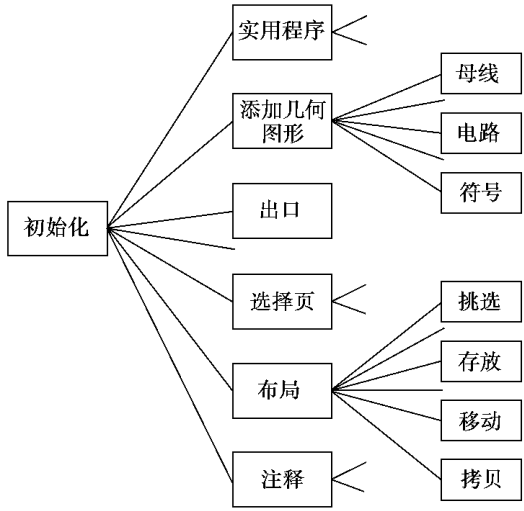
状态树由 OLIE 两个独立树组成：一个主树和一个辅助树。主树包含了在用户交互期间正常遇到的状态，所有主树状态都相互排斥，用户只能一次执行一个主命令。辅助树处理用户的中断，丝毫不失掉主树的上下文含义。例如，OLIE 有扫视命令和缩放命令，这两个命令在任何时候均可选用，即使在其他操作中间也可使用。扫视和缩放命令均保持这种控制，直至用户退出为止，仅当此时控制才返回到主树。

OLIE 开发的状态树的一个特征是菜单组织没有反映到状态结构之中，这就消除了在其

他方法中常常遇到过的问题：一系列“退出”或“出口”命令需要改变上下文含义。

2. 主树

OLIE 的主树的部分展示在图解 14-15 中。该树是基于在每种命令内在功能上的细分而组织成的。

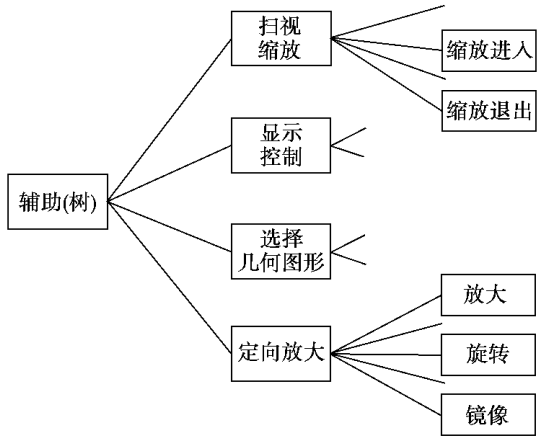


图解 14-15　主状态树

图解 14-15 中仅显示了某些状态。当程序开始时，就进入初始化状态，显示固定菜单，执行初始化，并把控制转换到选择页状态，允许用户编辑初始页。实用程序子状态建立报表、合成、绘图或创建电力分析程序所需要的文件。添加几何图形子状态建立单线图的项。出口状态查询用户已经完成保存了的任何变化。出口命令可由用户任何时候选取，因为任何命令均可以用出口函数，从当前状态到出口状态路径的状态树中清除掉。布局子状态执行编辑操作，诸如挑选、存放、移动和拷贝。注释子状态弹出正文表单，键入电力参数值。

3. 辅助树

图解 14-16 展示出辅助树，它包含了可以中断其他命令的状态命令。扫视-缩放用于改



图解 14-16　辅助状态树

变单线图的观察点；显示控制改变显示的选项，如表格或正文的可视性。选择几何图形提供控制单线图的选择元素，定向放大用于改变元素的位置或选择元素的尺寸。

4. 低层操作

对象由分析模型直接实现，在面向对象语言中使用关联对象直接地实现关联。相同的对象被存储在内存和数据库中（屏蔽子系统使用内存高速缓存的数据库）。因为保持映像清晰是很重要的，所以对象不优化。大多数操作直接从动态模型中实现，从分析直接映像到设计是交互编辑器共同的事，通常由松散的操作收集组成，即是直接的在对象模型上的行为。在需求分析期间，为使用户命令能够发出，试图避免依赖性是重要的，这样就可能直接地从分析映像到设计。

• 实现

通常，在实现中不总是清楚子系统之间是如何相互作用的。每个子系统期望找到公共的封装操作的软件生产率工具。OLIE 可以立即应用于所有子系统。在实现中，经常碰到资源需求、使用上约束或与其他子系统在操作哲理上冲突。例如，图形子系统、菜单子系统、状态树子系统之间重叠。为了使每个子系统都以原始设计方式工作，需要各自排斥访问用户的输入输出。为此我们对所有三个子系统在一起工作进行了修改。当外部约束强加在体系结构和实现上时，这样的困难是很难避免的。

OLIE 包含了大约 10 万行代码，估计采用面向对象方法大约减少设计和实现时间为一个人年，而且结果系统容易维护。对象模型技术（OMT）可以帮助设计者解决所遇到的许多问题，在此状态树简化了用户接口的模型。

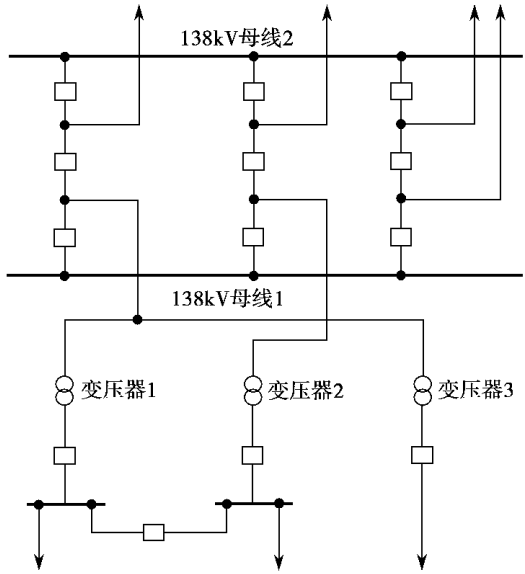
• 背景

“单线图”是电力系统的图解式图样，所有电力工程师都采用这种简洁、标准的表示法。只要从所表示的三个模型系统的其中一个模型，单线图就可获取它们的名字。标准符号用来表示电力系统的部件，诸如变压器、线路断路器、发电机、导线和开关。电路和母线连接到标准符号，电路和母线之间的鉴别是基于电压下降，电路与可测量电压下降相连接（特别在电力系统上存在短路的情况），并且通常是用导线、电缆或传输线实现。母线与不明显的电压下降相连接，它一般是用具有大型截面的铝、铜棒或导管来实现。

图解 14-17 展示了部分单线图的样本，表示了两个高压（138kV）母线。图中顶部箭头表示电力电源，诸如多用途电源（实用电源）。底部箭头表示负载。方块表示线路断路器，变压器用锯齿线表示，小黑圆点把重叠线与连接线区分开来，四个独立的电源通过九个线路断路器连接到两个高压母线。用选择其中一个线路断路器打开而另一个关闭的方式，电力系统操作符能够用几个方式配置以供应三个变压器的电力。在构造附加电路期间，允许断开一条母线的连接。例如，在变压器 1 和变压器 2 的负载能够关闭连接两个辅助母线的断路器的任何一个变压器进行服务和供应。

开发 OLIE 的原动力是得益于最近高级计算机技术的发展。不同的资历和素质的工程师都希望替代 40 年前所开发的旧的分析软件，这些软件均操作在批处理方式，并且要求提交输入的卡片映像。确定仿真的较自然的方法是在屏幕上绘制一个图并在图上注释出绘制的期望设计条件。这种新方法允许交互作用的菜单驱动，并在执行期间尽可能的与仿真

程序交互作用。例如，工程师能够检查仿真的收敛域，或者工程师可以改变计算过程的设计条件。



图解 14-17　单线图样本

*　*　*　*　*　*　*

OLIE 是设计电力分布式系统的计算机辅助设计 (CAD) 工具。OLIE 支持电力工程师所能接受的标准表示的单线图绘制。单线图是二维图，它表示电力设备之间的连接。开发 OLIE 基本目标是提供一个完美的用户接口和获取在数据库中单线图的内容，以便该内容信息能用于其他目的。OLIE 软件开发工作取决于使用某个软件子系统的策略。

OLIE 有一个丰富的对象模型，因为 OLIE 中存有许多电力部件的规格说明和所有数据必须要存储在数据库中。动态模型也是很有意义的，因为一个完美的用户接口是很重要的。功能模型对 OLIE 是不重要的。OLIE 系统设计中的大量工作是确定建立几个子系统的需求，对象设计和实现对这个应用来说是简单明了的。

14.2　试用 C++ 语言编制学生地址簿 (Student Address Book——SAB) 程序。具体要求如下：

⑴ 用窗口显示一个学生的基本信息，比如学生名字、年龄、地址等，

⑵ 学生地址簿允许用户添加、修改学生的基本信息；

⑶ 允许用户浏览学生地址簿；

⑷ 允许用户创建新的学生地址簿，打开已有的学生地址簿，存储并修改学生地址簿；

⑸ 允许学生地址簿包含超过 100 个学生以上的基本信息。

答：我们知道开发一个软件的生命周期通常包括以下 6 个步骤：

1. 需求分析；
2. 规格说明 (specification)；
3. 设计；
4. 编码 (coding)；
5. 测试与诊断；

6. 运行。

开发学生地址簿 (SAB) 也不例外，鉴于篇幅，我们只对前 4 个步骤加以解答，第 5，第 6 个步骤留给读者自己完成。

1. 学生地址簿 (SAB) 的需求分析

我们用 C++ 语言开发的 SAB 的需求必须有效地存储学生的名字、年龄、电话号码和地址等信息。该程序应允许操作多个学生地址簿，并要求在任何一个地址簿中可以存储 100 个以上学生的基本信息。

2. SAB 规格说明

根据上述需求，我们必须产生完整、精确、一致地规格说明。即：

- 用窗口显示一个学生的基本信息，比如学生名字、年龄、地址等；
- 学生地址簿允许用户添加、修改学生的基本信息；
- 允许用户浏览学生地址簿；
- 允许用户创建新的学生地址簿，打开已有的学生地址簿，存储并修改学生地址簿；
- 允许学生地址簿包含超过 100 个学生以上的基本信息。

3. 设计

面向对象程序设计 (OOP) 一个关键优点是代码可重用，所以在设计阶段必须对可重用对象倍加注意进行设计。

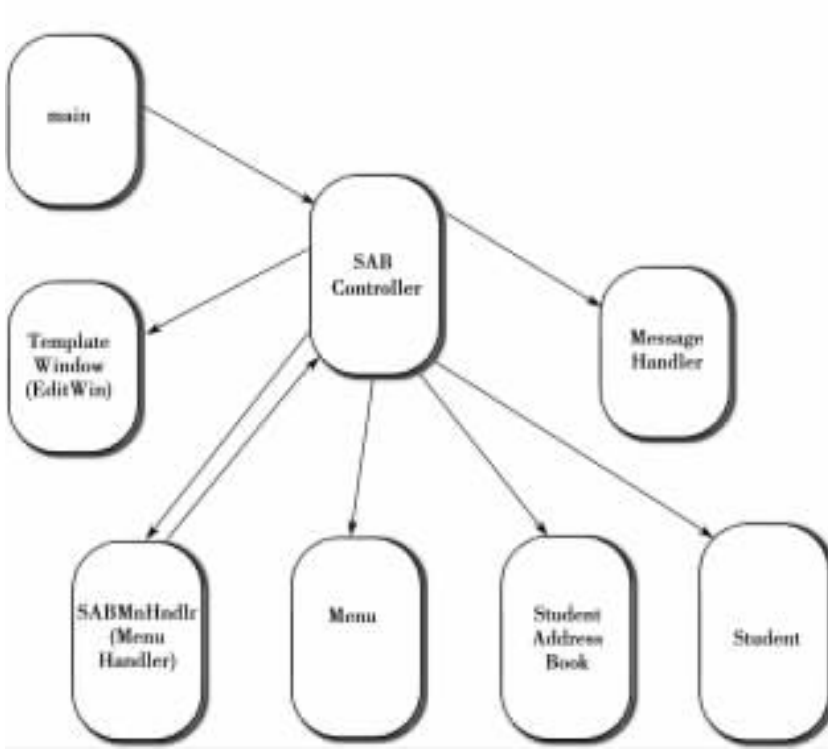
通常对象是可以分层次的。在不同层次的对象蕴含着相关的继承关系。最低层是系统层，对象在系统层中是独立于应用的，比如 GUI 对象就在系统层。第二层是应用领域层，在这层的对象位于系统层之上，最靠近相关的程序特定类型。最后一层是程序层，这层的对象与特定的程序相关联，这些程序不会在其他程序中可重用。

对 SAB 来说，我们首先要标识特定的对象。图解 14-18 表示了 SAB 程序的对象图，图解 14-19 表示了四个分类三个层次图。注意：在这两个图中都没有把标准的 GUI 对象示出。

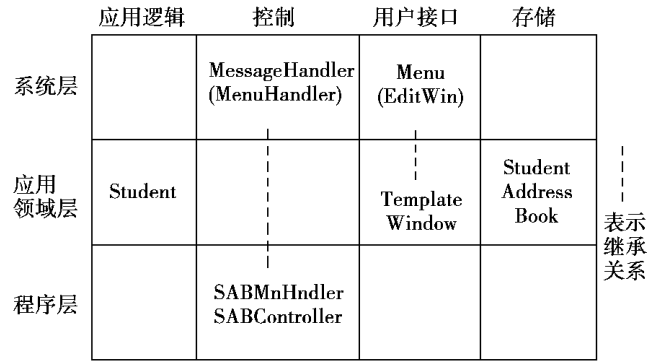
SABController 管理所有的对象。Menu 对象有三个：File、Edit 和 Move。显示和控制这些菜单对象。菜单选择事件用 SABMnHndler 处理，是 MenuHandler 的子孙。当一个菜单被选上时，SABMnHndler 就通知 SABController。菜单选择过程用 SABController 通过从属对象协助实现。图解 14-20 是程序的菜单选择。

我们对每个菜单选择定义一个 SABController 函数，编辑和浏览 Student 对象的操作，采用应用逻辑对象 StudentAddressBook 实现。这个对象知道如何把已打开的 Student 数据存储到外部文件中去。单一的 Student 数据用用户接口对象显示，是 TemplateWindow，EditWin 的子孙。所有这些从属对象都是 SABController 的私有成员，下面是其接口：

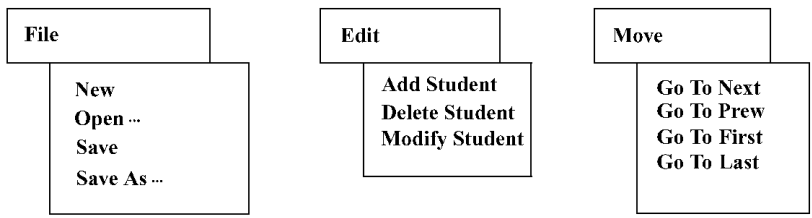
```
class SABController {
  public:
    SABController();
    void Start();
    void NewAddrBook();          //处理文件菜单
    void OpenAddrBook();
    void SaveAddrBook();
    void SaveAsAddrBook();
```



图解 14-18　SAB **程序的对象图**



图解 14-19　SAB **的对象四种分类和三个层次图**



图解 14-20　SAB **程序的菜单选择**

```
    void AddStudent();          //处理编辑菜单
    void DeleteStudent();
    void ModifyStudent();
    void GotoNext();            //处理移动菜单
    void GotoPrev();
    void GotoFirst();
    void GotoLast();
  private:
    Menu fileMenu;
    Menu editMenu;
    Menu moveMenu;
    SABMnHndlr menuHandler;
    MessageHandler msgHandler;
    TemplateWindow oneStudentDisplay;
    StudentAddressBook stuaddrBook;
    void SetupMenu();
  };
```

下面我们挑选几个函数，看看它们是如何实现的。一个是 SABController **定义的构造函数。这个构造函数初始三个由每个正文指定的菜单对象。下面是** SABController **类的构造函数：**

```
SABController::SABController()
    :fileMenu("File"),editMenu("Edit"),moveMenu("Move")
{
}
```

该函数调用的是构造函数中的数据成员，而不是函数体中的数据成员。

在 Start **函数中建立从属的** Menu **和** TemplateWindow **对象。在私有函数** SetupMenu **建立菜单选项和相关的菜单处理**

Open **函数的实现如下：**

```
void  StudentAddressBook::Open(char * filename){
      addrBookFile.open(filename,ios::in);
      strcpy(adBkFilename,filename);
      char *value[10],tmpStr[255];
      int entrySize, attrSize,i,j;
      addrBookFile >> entrySize;      //读控制信息
      addrBookFile >> attrSize;
      addrBookFile.getline(tmpStr,255);      //读传送的第二行信息
      entry Count = entrySize;                //读数据
for(i = 0;i < entrySize;i = i + 1){
    //读一个学生信息
    for(j = 0;j <= attrSize - 1;j = j + 1){
```

```
        addrBookFile.getline(tmpStr,255);
        value[j] = new char[strlen(tmpStr) + 1];
        strcpy(valne[j],tmpStr);
    }
    entry[i] = new Student;
    entry[i] -> SetValue(value);
}
addrBookFile.close();
```

其他的输入/输出函数都与 Open 函数类似，在此不多叙了。

下面讨论一下添加一个新的学生的操作。要添加一个学生 StudentAddressBook 必须寻找一个打开槽并插入数据，Student 的数组指针 (entry) 用 StudentAddressBook 来操作数据。

```
void StudentAddressBook::Add(Student * student){
    if(entryCount < SAB_MAXSIZE){              //仍有空间添加
      entry[entryCount] = student;
      entryCount = entryCount + 1;
      changed = 1;
}
```

Move 操作是非常清楚的，我们只简单讨论一下 Next 函数，其他函数都是类似的。

Next 函数首先检查一下是否存在下一个 Student 对象 (这与当前选择的 Student 对象有关)，然后调整当前变量，返回指向下一个 Student。这样 Next 函数写成：

```
Student * StudentAddressBook::Next(){
    if(entryCount <= 0 || current == entryCount - 1)
    return 0;//错误,无数据项或无前一个
    else{
      current = current + 1;
      return entry[current];
  }
}
```

最后，我们来讨论一下 StudentAddressBook 对象，这是该 SAB 程序的主框架，它管理 SAB 的能力表现在添加、删除、修改数据，以及浏览存储在文件中的地址簿信息。其中 DataChanged，HasData 等函数用来让客户进行对象查询，并在如果新数据已经添加的话，请求 StudentAddressBook 是否将数据进行更新。它的接口如下：

```
const int SAB_MAXSIZE = 100;
class StudentAddressBook{
  public:
    StudentAddressBook();
    void Clear();
    int HasData();                    //如果 SAB 包含了数据,返回真
    int DataChanged();                //如果 SAB 中数据修改了,返回真
```

```
    int HasFile();                  //如果 SAB 已有一个相关文件,返回真
    void Open(char * filename);     //打开文件的文件名并装入数据
    void Save();                    //存储数据到当前相关文件中
    void Save(char * filename);     //把数据存入文件名中
    void Add(Student * student);    //添加一个学生到 SAB 中
    Student * Next();               //在列表中返回下一个学生
    Student * Prev();               //返回前一个学生
    Student * First();              //返回第一个学生
    Student * Last();               //返回最后一个学生
  private:
    fstream stuaddrBookFile;
    char stuadBkFilename[50];
    Student * entry[SAB_MAXSIZE];
    int entryCount;
    int current;
    int dataChanged;
};
```

4. **编码** (coding)

学生地址簿 (SAB) C++ **程序如下：**

1)SABMain.cpp

```
#include"SAdBkCtl.h"
void main(){
    SABController controller;
    controller.Start();
}
```

2)SAdBkCtl.h

```
#ifndef   _SADBKCTL_H
#define   _SADBKCTL_H
#include "SABMnHd.h"
#include "Student.h"
#include "StuAddrBook.h"
#include "TplWndw.h"
const int SAB_MAXENTRY=10;
class SABController{
  public:
    SABController();
    void Start();
    void NewAddrBook();             //文件菜单处理
    void Open AddrBook();
```

```
    void Save AddrBook();
    void Save AsAddrBook();
    void AddStudent();              //编辑菜单处理
    void DeleteStudent();
    void ModifyStudent();
    void GotoNext();                //移动菜单处理
    void GotoPrev();
    void GotoFirst();
    void GotoLast();
  private:
    Menu fileMenu;
    Menu editMenu;
    Menu moveMenu;
    SABMnHndlr              menuHandler;
    MessageHandler          msgHandler;
    TemplateWindow          oneStudentDisplay;
    StudentAddressBook      stuaddrBook;
    void SetupMenu();
  };
#endif
```

3)SAdBCtl.cpp

```
#include "SAdBkCtl.h"
SABController::SABController()
      :fileMenu("File"),editMenu("Edit"),MoveMenu("Move"){
}
void SABController::Start(){
      Student oneEntry;
      char *attributes[SAB_MAXENTRY];
      SetupMenu();
      //初始化显示窗口并列出它的标号
      oneEntry.GotAttr(attributes);
      oneStudentDisplay.Init(100,75,350,250);
      oneStudentDisplay.setLabel(attributes);
      oneStudentDisplay.setTitle("Student Information");
      msgHandler.Start();
}
void SABController::SetupMenu(){
      //设置本身作为 menuHandler 之主
      menuHandler.SetOwner(this);
```

```
        //添加单个菜单选项
        fileMenu.AddItem("New",             menuHandler,1);
        fileMenu.AddItem("Open...",         menuHandler,2);
        fileMenu.AddItem("Save",            menuHandler,3);
        fileMenu.AddItem("Save as...",      menuHandler,4);
        editMenu.AddItem("Add Student",     menuHandler,5);
        editMenu.AddItem("Delete Student",  menuHandler,6);
        editMenu.AddItem("Modify Student",  menuHandler,7);
        moveMenu.AddItem("Go To Next",      menuHandler,8);
        moveMenu.AddItem("Go To Prev",      menuHandler,9);
        moveMenu.AddItem("Go To First",     menuHandler,10);
        moveMenu.AddItem("Go To Last",      menuHandler,11);
}
void SABController::NewAddrBook(){
        YesNoBox  pronpt;
        //如果存在数据,就提示用户
        if (stuaddrBook.Datachanged()){
            //如果用户请求,就存储当前数据
            if (prompt.Ask)
                "Data have been changed, but not yet saved. Save new?"))
              //用户肯定(YES),则存到 SAB 中
              stuaddrBook.Save();
            }
        else
              stuaddrBook.Clear();
}
void SABController::OpenAddrBook(){
        YESNoBox prompt;
        char filename [50];
        FileBox fileBox;
        if(stuaddrBook.Datachanged()){//如果用户请求,就存未被存储修改的数据
            if(prompt.Ask)
              "Data have been changed, but not saved yet. Save now?"))
              //用户肯定(YES),则存到 SAB 中
              stuaddrBook.Save();
            }
        //打开文件名
        strcpy(filename,fileBox.GetOpenFileName());
       if(strepy(filename,"")!=0 //名字已给出,所以能打开它 否则撤销,什么也不做
```

```
            stuaddrBook.Open(filename);
}
void SABController::SaveAddrBook(){
      char filename[50];
      FileBox fileBox;
      //已被修改的数据,要存吗?
      if (stuaddrBook.DataChanged())
          if (stuaddrBook.HasFile())  //已有了相关文件
            stuaddrBook.Save();
          else{
            strcpy(filename, fileBox.GetSaveFileName());
            if(strcpy(filename."")!=0)//名字已给出,所以存储它;
                                      //否则撤销,什么也不做
              stuaddrBook.Save(filename);
          }
}
void SABController::Save AsAddrBook(){
      char filename[50];
      File Box fileBox;
      OKBox msg;
      //已有的数据,要存吗?
      if(stuaddrBook.HasData()){    //给出要存的文件名
        strcpy(filename, fileBox.GetSavefileName());
       if(strepy(filename,"")!=0)    //名字给定,所以存它 否则撤销,什么也不做
        stuaddrBook.Save(filename);
      }
      else
        msg.Display("No data to save");
}
void SABController::DeleteStudent(){}
void SABController::ModifyStudent(){}
void SABController::AddStudent(){
      char *attributes [SAB_MAXENTRY];
      Student *oneEntry;
      oneStudentDisplay.GotData(attributes);
      oneEntry ->SetData(attributes);
      stuaddrBook.Add(oneEntry);
      oneStudentDisplay.Clear();
```

```
}
void SABController::GotoNext(){
      Student *oneEntry;
      char *values[SAB_MAXENTRY];
      if (oneEntry=stuaddrBook.Next()){
            oneEntry->GetValue(values);
            oneStudentDisplay.SetData(values);
      }
      else{      //无下一个数据
            OkBox msg;
            msg.Display("No More Data");
      }
}
void SABController::GotoPrev(){
      Student *oneEntry;
      char *values[SAB_MAXENTRY];
      if (oneEntry=stuaddrBook.Prev()){
          oneEntry->GetValue(values);
          oneStudentDisplay.SetData(values);
      }
      else{      //无前一个数据
        OkBox msg;
        msg.Display("No More Data");
      }
}
void SABController::GotoFirst(){
      Student *oneEntry;
      char *values[SAB_MAXENTRY];
      if (oneEntry=stuaddrBook.First()){
          oneEntry->GotValue(values);
          oneStudentDisplay.SetData(values);
      }
      else{      //没有数据
        OkBox msg;
        msg.Display("No Data");
      }
}
void SABController::GotoLast(){
```

```
      Student * oneEntry;
      char * values [SAB_MAXENTRY];
      if (one Entry = stuaddrBook.Last()){
          oneEntry -> GetValue(values);
          oneStudentDisplay.SetData(values);
      }
      else{      //没有数据
        OkBox msg;
        msg.Display("No Data");
      }
}
```

4)SABMnHd.h

```
#ifndef  _SADBKMNHD_H
#define  _SADBKMNHD_H
#include "GUIobj.h"
class SABController;  //向前引用
class SABMnHndlr:public MenuHandler{
    public:
        void SetOwner(SABController * teacher);
        void MenuFunc1();      //New
        void MenuFunc2();      //Open...
        void MenuFunc3();      //Save
        void MenuFunc4();      //Save as...
        void MenuFunc5();      //Add Student
        void MenuFunc6();      //Delete Student
        void MenuFunc7();      //Modify Student
        void MenuFunc8();      //Go To Next
        void MenuFunc9();      //Go To Prev
        void MenuFunc10();     //Go To First
        void MenuFunc11();     //Go To Last
    private:
        SABController * owner;
};
#endif
```

5)SABMnHd.cpp

```
#include "SAdBkCtl.h"
#include "SABMnHd.h"
void SABMnHndlr::SetOwner (SABController * teacher){
        owner = teacher;
```

```
}
void SABMnHndlr::MenuFunc1(){        //New
        owner->NewAddrBook();
}
void SABMnHndlr::MenuFunc2(){        //Open...
        owner->OpenAddrBook();
}
void SABMnHndlr::MenuFunc3(){        //Save
        owner->SaveAddrBook();
}
void SABMnHndlr::MenuFunc4(){        //Save as...
        owner->SaveAsAddrBook();
}
void SABMnHndlr::MenuFunc5(){        //Add Student
        owner->AddStudent();
}
void SABMnHndlr::MenuFunc6(){        //Delete Student
        owner->DeleteStudent();
}
void SABMnHndlr::MenuFunc7(){        //Modify Student
        owner->ModifyStudent();
}
void SABMnHndlr::MenuFunc8(){        //Go To Next
        owner->GotoNext();
}
void SABMnHndlr::MenuFunc9(){        //Go To Prev
        owner->GotoPrev();
}
void SABMnHndlr::MenuFunc10(){       //Go To First
        owner->GotoFirst();
}
void SABMnHndlr::MenuFunc11(){       //Go To Last
        owner->GotoLast();
}
6)Tplwndw.h
#ifndef  _TPLWNDW_H
#define  _TPLWNDW_H
#include <string.h>
```

```
#include "GUIobj.h"
class TemplateWindow:public EditWin{
    public:
      void GetData(char *value[]);  //获得数据项列表
      void SetData(char *value[]);  //显示有正确提示的数据
      void SetLabel (char *tmpl[]); //设定属性项的提示列表
    private:
      char *label[10];
      int labelSize;
};
#endif
```

7)TplWndw.cpp

```
#include "TplWndw.h"
void TemplateWindow::SetLabel(char *tmpl[]){
      int i=0;
      while (tmpl[i]!=0){            //not NULL
          label [i]=new char[20];
          strcpy(label[i],tmpl[i]);
          i=i+1;
      }
      labelSize=i;
}
void TemplateWindow::SetData(char *value[]){
      char str[40]
      Clear();
      for (int i=0;i<labelSize; i=i+1){
          InsertLine (2*i,label[i]);//显示标号
          strcpy(str,"      ");
          strcat(str,value[i]);
          InsertLine(2*i+1,str);   //显示值
      }
}
void TemplateWindow::GetData(char *value[]){
      for(int i=0;i<labelSize;i=i+1){
          value[i]=new char[40];
          strcpy(value[i],ReadLine(i));
      }
}
```

8)Student.h

```
#ifndef  _STUDENT_H
#define  _STUDENT_H
#include <stdlib.h>
#include <string.h>
class Student{
      public:
          void SetValue (char *value[]);     //对每个属性指定数据
          void GetAttr(char *attr[]);        //返回设定的属性名
          void GetValue(char *value[]);      //返回设定的属性值
          void NumAttr();                    //返回属性个数
      private:
          char  name[20];
          int   age;
          char  address[40];
          char  phone[14];
};
#endif
```

9)Student.cpp

```
#include "Student.h"
void Student::GetValue(char *value[]){
      value[0]=new char [strlen(name)+1];
      strcpy(value[0],name);
      value[1]=new char[3];
      itoa(age,value[1],10);
      value[2]=new char [strlen(address)+1];
      strcpy(value[2],address);
      value[3]=new char [strlen(phone)+1];
      strcpy(value[3],phone);
}
void Student::GetAttr(char *attr[]){
      attr[0]="Name:";
      attr[1]="Age:";
      attr[2]="Address:";
      attr[3]="Phone:";
}
void Student::SetValue(char *value[]){
      Strcpy(name,value[0]);
      age=atoi(value[1]);
      strcpy(address,value[2]);
```

```
      strcpy(phone,value[3]);
}
int Student::NumAttr(){
      return 4;
}
```

10)StuAddrBook.h

```
#ifndef    _STUADDRBOOK_H
#define    _STUADDRBOOK_H
#include   <fstream.h>
#include   "Student.h"
#include   "GUIobj.h"
const int SAB_MAXSIZE=100;
class StudentAddressBook {
      public:
          StudentAddressBook();
          void Clear();
          int HasData();                    //如果包含了数据,返回真
          int DataChanged();                //如果数据修改了,返回真
          int HasFile();                    //如果已有了相关的文件,返回真
          void Open (char *filename);       //打开该文件并装入数据
          void Save();                      //将数据存到当前相关文件中
          void Save (char *filename);       //把数据存到文件名中
          void Add(Student *student);       //把一个学生添加到学生地址簿 SAB 中
          Student *Next();                  //返回该列表中下一个学生
          Student *Prev();                  //返回该列表中前一个学生
          Student *First();                 //返回第一个学生
          Student *Last();                  //返回最后一个学生
      private:
          fstream       stuaddrBookFile;
          char          stuadBkFilename[50];
          Student       *entry[SAB_MAXSIZE];
          int           entryCount;
          int           current;
          int           changed;
};
#endif
```

11)StuAddrBook.cpp

```
#include "StuAddrBook.h"
#include "Student.h"
```

```
StudentAddressBook::StudentAddressBook(){
        Clear()
}
void StudentAddressBook::Clear(){
    entryCount =0;
    current    = -1;
    changed    =0;
    strcpy(stuadBkFilename,"");
}
int StudentAddressBook::HasData(){
    if(entryCount >0)
        return 1;
    else
        return 0;
}
int Student AddressBook::DataChanged(){
    return changed;
}
int StudentAddressBook::HasFile(){
    if(strcmp(stuadBkFilename,"")!=0)
        return 1;
    else
        return 0;
}
void StudentAddressBook::Open (char * filename){
    char * value [10],tmpStr [255];
    int entrySize, attrSize,i,j;
    stuaddrBookFile.open(filename,ios::in);
    strcpy(stuadBFilename,filename);
    stuaddrBookFile >>entrySize;//读控制信息
    StuaddrBookFile >>attrSize;
    entryCount =entrySize;
    stuaddrBookFile.getline(tmpStr,255);//读第二行信息
    //读数据
    for(i =0;i <entrySize;i =i +1){
      //读一个学生信息
    for(j =0;j <attrSize;j =j +1){
        stuaddrBookFile.getline(tmpStr,255);
```

```
            value[j] = new char [strlen(tmpStr) + 1];
            strcpy(value[j],tmpStr);
        }
            entry[i] = new Student;
            entry[i] -> SetValue(value);
        }
        stuaddrBookFile,close();
}
void StudentAddressBook::Add(Student * student){
        if(entryCount < SAB_MAXSIZE){  //仍有空间可添加数据
            entry[entryCount] = student;
            entryCount = entryCount + 1;
            changed = 1
        }
}
void StudentAddressBook :: Save(){
        int attrSize,i,j;
        char * value[10];
        if (HasFile()){//存储已有的文件
            stuaddrBookFile.open(stuadBFilename,ios :: out);
            attrSize = entry[0] -> NumAttr();
            stuaddrBookFile << entryCount << "h n";//存储控制信息
            stuaddrBookFile << attrSize << "h n";
            for(i = 0;i < entryCount;i = i + 1){
                entry[i] -> GetValue(value);
                //读一个学生信息
                for(j = 0;j < attrSize;j = j + 1)
                    stuaddrBookFile << value[j] << "hn";
                delete entry[i];
                }
                stuaddrBookFile.close();
                changed = 0;
            }
}
void StudentAddressBook::Save(char * filename){
        int attrSize,i,j;
        char * value[10];
        stuaddrBookFile << entryCount << "hn";  //存储控制信息
        stuaddrBookFile << attrsize << "hn";
```

```
        for(i=0;i<entryCount;i=i+1){
            entry[i]->GetValue(value);
            //读一个学生信息
            for(j=0;j<attrSize;j=j+1)
                stuaddrBookFile<<value[j]<<"\n";
            delete entry[i];
        }
        stuaddrBookFile.close()
        changed=0;
    }
    Student *StudentAddressBook::Next(){
        if(entryCount<=0||current==entryCount-1)
            return 0;  //错误,无数据或无前一个数据
        else{
            current=current+1;
            return entry[current];
        }
    }
    Student *StudentAddressBook::Prev(){
        if (entryCount<=0 || current==0)
            return 0;    //错误,无数据或无前一个数据
        else{
            current=current-1;
            return entry[current];
        }
    }
    Student *StudentAddressBook::First(){
        if (entryCount<=0)
            return 0;    //错误,无数据
        else{
            current=0;
            return entry[current];
        }
    }
    Student *StudentAddressBook::Last {
        if (entryCount<=0)
            return 0;    //错误,无数据
        else{
```

```
            current = entryCount - 1;
            return entry [current];
        }
}
```

第二部分　上 机 实 践

这部分内容包括了函数应用、类与对象、数组、指针和字符串、多态性、流类库的输入/输出等，一共 26 道上机练习题。读者上机的环境既可以采用 VC++，也可以采用Borland C++。具体的上机实践环境读者可参照相应的参考书，比如 VisualC++开发应用环境。

实践 1　产生列表表格

```
//实践 1. cpp
#include <iostream>
#include <iomanip>   //引入 setw()函数
using namespace std;

int main(){
    cout <<2000 <<setw(8) <<188 <<endl
         <<2001 <<setw(8) <<8380 <<endl
         <<2002 <<setw(8) <<11800 <<endl
         <<2003 <<setw(8) <<18320 <<endl;
    return 0;
}
```

实践 2　递减赋值运算

```
//实践 2. cpp
#include <iostream>
using namespace std;

int main(){
    int x=20;
    cout <<x <<endl;        //x 是 20
    x *=2;                  //x 变成 40
    cout <<x-- <<endl;      //先显示 x,然后再递减 x
    cout <<x <<endl;        //x 现在为 39
    return 0;
}
```

实践 3　华氏温度与摄氏温度相互转换

```
//实践 3. cpp
#include <iostream>
using namespace std;

int main(){
    int response;
    double temper;
    cout <<"\n 键入 1 是把华氏转为摄氏,"
         <<"\n 键入 2 是把摄氏转为华氏:";
    cin >>response;
    if(response==1){
        cout <<"请输入华氏温度:";
        cin >>temper;
        cout <<"摄氏温度是"<<5.0/9.0* (temper-32.0);
    }
    else{
        cout <<"请输入摄氏温度:";
        cin >>temper;
        cout <<"华氏温度是"<<9.0/5.0* temper+32.0;
    }
    cout <<endl;
    return 0;
}
```

实践 4　只有四则运算的计算器

```
//实践 4.cpp
#include <iostream>
using namespace std;

int main(){
    double n1,n2,answ;
    char oper, ch;
    do{
    cout <<"\n 请输入第一个数,操作符,第二个数:";
    cin >>n1 >>oper >>n2;
```

```
    switch (oper){
        case '+':answ=n1+n2;  break;
        case '-':answ=n1-n2;  break;
        case '*':answ=n1*n2;  break;
        case '/':answ=n1/n2;  break;
        default : answ=0;
    }
    cout<<"Answer="<<answ;
    cout<<"\n Do another (Enter 'y'or 'n')?";
    cin >>ch;
    }while(ch!='n');
    return 0;
}
```

实践 5　用结构 (struct) 来计算房子的体积

```
//实践 5. cpp
#include <iostream>
using namespace std;

struct Range{
    int feet;
    float inches;
};
struct Volume{
    Range length;
    Range width;
    Range height;
};

int main(){
    float l,w,h;
    Volume room={{18,3.5},{12,6.25},{8,1.75}};
    l=room.length.feet+room.length.inches/12.0;
    w=room.width.feet+room.width.inches/12.0;
    h=room.height.feet+room.height.inches/12.0;
    cout<<"Volume="<<l*w*h<<"cubic feet\n";
    return 0;
}
```

实践 6　用函数计算圆的面积

```
//实践 6.cpp
#include <iostream>
using namespace std;
float circarea (float radius);
int main(){
    double radi;
    cout <<"\n 请输入圆的半径:";
    cin >> radi;
    cout <<"圆的面积是" <<circarea(radi) <<endl;
    return 0;
}
//计算面积的函数
float circarea(float r){
    const float PI = 3.14159F;
    return r * r * PI;
}
```

实践 7　用内联函数完成磅与公斤的互换

```
//实践 7. cpp
#include <iostream>
using namespace std;

//用 lbstokg()内联函数完成磅与公斤的转换
inline float lbstokg(float pounds){
    return  0.453592 * pounds;
}

int main(){
    float lbs;
    cout <<"\n 请用磅来输入你的体重:";
    cin >> lbs;
    cout <<"按公斤计算你的体重是:" <<lbstokg(lbs)
         <<endl;
    return 0;
}
```

实践 8　采用引用传递方式查找实数的整数和小数部分

```
//实践 8. cpp
#include <iostream>
using namespace std;
int main(){
    void intfrac(float,float&,float&);                    //说明浮点变量
    float number, intpart, fracpart;
    do{
      cout <<"\n 请你输入一个实数:";                      //用户输入
      cin >>number;
      intfrac(number,intpart,fracpart);                   //寻找 int 和 frac
      cout <<"整数部分是" <<intpart                       //打印整数和小数
           <<",小数部分是 <<fracpart <<endl;
      }while(number!=0.0);                                //当 0.0 时跳出循环
    return 0;
}
//用 intfrac()函数寻找实数的整数和小数部分
void intfrac(float n,float & intp,float & fracp){
    long temp = static_cast<long>(n);                     //转换为长整数
    intp = static_cast<float>(temp);                      //返回为浮点数
    fracp = n - intp;                                     //减去整数部分
}
```

实践 9　把整数数据类型模型化为类

```
//实践 9. cpp
#include <iostream>
using namespace std;

class Int{                                  //与 int 是不相同的
    private:
        int i;
    public:
        Int(){                              //创建一个 Int
          i=0;
        }
        Int(int ii){                        //创建并初始化一个 Int
```

```
        i = ii;
      }
      void add(Int i2;Int i3){  //将两个 Int 相加
        i = i2.i + i3.i;
      }
      void display(){           //显示一个 Int
        cout << i;
      }
};

int main(){
    Int Int1(8);                //创建并初始化一个 Int
    Int Int2(17);               //创建并初始化一个 Int
    Int Int3;                   //创建一个 Int

    Int3.add(Int1,Int2);        //将两个 Int 相加
    cout << "\nInt3 = ";
    Int3.display();             //显示其结果
    return 0;
}
```

实践 10　将 C 串反转

```
//实践 10.cpp
#include <iostream>
#include <cstring>              //引入 strlen()函数
using namespace std;

int main(){
    void reversit(char []);     //原型
    const int MAX = 80;         //数组长度
    char str[MAX];              //串
    cout << "\n 请键入一个串:";  //从用户那里获得一串
    cin.get(str,MAX);
    reversit(str);              //将串反转
    cout << "反转后的串是:";    //显示反转后的串
    cout << str << endl;
    return 0;
}
//用 reversit()函数反转一个串
```

```
void reversit(char s[]){
    int len=strlen(s);              //查串的长度
    for (int j=0;j<len/2;j++){
        char temp=s[j];             //用第二半的字符与第一个的每个字符进行交换
        s[j]=s[len-j-1];
        s[len-j-1]=temp;
    }
}
```

实践11　用串作为数据的雇用对象

```
//实践11.cpp
#include <iostream>
#include <string>
using namespace std;

class employee{
    private:
        string name;
        long number;
    public:
        void getdata(){                 //从用户获得数据
            cout<<"\n请输入你的名字:";
            cin>>name;
            cout<<"\n请输入你的雇用号:";
            cin>>number;
        }
        void putdata(){                 //显示数据
            cout<<"\n名字:"<<name;
            cout<<"\n雇用号:"<<number;
        }
};

int main(){
    employee emparr[100];               //雇员数组
    int n=0;                            //多少个雇员
    char ch;                            //用户回答
    do {
        cout <<"\n请输入雇用号的数据"<<n+1;
```

```
        emparr[n++].getdata();
        cout<<"\n是否键入另一个雇员(y/n)?";
        cin>>ch;
        }while(ch!='n');
    for(int j=0;j<n;j++){          //显示数组的数据
        cout<<"\n雇员号"<<j+1;
        emparr[j].putdata();
    }
    cout<<endl;
    return 0;
}
```

实践 12　用运算符‘+=’重载来连接两个串

```
//实践12.cpp
#include <iostream>
#include <cstring>                 //引入strcpy(),strlen()函数
using namespace std;
#include <process.h>               //引入exit()函数

class String {                     //用户定义串类型
    private:
        enum{SZ=80};               //String对象的长度
        char str[SZ];              //保留一个C-串
    public:
        String(){                  //无参的构造函数
          strcpy(str,"");
        }
        String(char s[]){          //一个参数的构造函数
          strcpy(str,s);
        }
        void display(){            //显示该String
          cout<<str;
        }
      String operator +=(String ss){  //把一个String加到这个串中,
                                      //其结果就在此串中
        if(strlen(str)+strlen(ss.str)>=SZ){
          cout <<"\nString溢出";
          exit(1);
```

```
        }
        strcat(str,ss.str);          //添加自变量串
        return String(str);          //返回 temp String
        }
};

int main(){
      String s1 = "Merry Christmas!"; //使用一个参数的串
      String s2 = "Happy new year!";  //使用一个参数的串
      String s3;                      //使用参数的串
      s3 = s1 + s2;                   //s1 加 s2,赋给 s3
      cout << "\n s1 = ";
      s1.display();          //显示 s1
      cout << "\n s2 = ";
      s2.display();          //显示 s2
      cout << "\n s3 = ";
      s3.display();          //显示 s3
      cout << endl;
      return 0;
}
```

实践 13　用重载‘ + ’运算符将两个时间 (time) 相加

```
//实践 13. cpp
#include <iostream>
using namespace std;

class time{
    private:
      int hrs, mins, secs;
    public:
      time():hrs(0),mins(0),secs(0)  //无参构造函数
          {}
      time (int h, int m, ints):hrs(h),mins(m),secs(s)
          {}                          //3 个参数构造函数
      void display(){
        cout << hrs << ":" << mins << ":" << secs;
      }
      time operator + (time t2){       //两个 time 相加
```

```
        int s = secs + t2.secs;          //加秒
        int m = mins + t2.mins;          //加分钟
        int h = hrs + t2.hrs;            //加小时
        if(s > 59){                      //如果秒溢出,进到分钟
          s -= 60;
          m ++;
        }
        if(m > 59){                      //如果分钟溢出,进到小时
          m -= 60;
          h ++;
        }
        return time(h,m,s);              //返回 temp 值
        }
}

int main(){
      time time1(5,59,59);               //创建两个 time 并初始化
      time time2(4,30,30);
      time time3;                        //创建另一个 time
      time3 = time1 + time2;             //两个 time 相加
      cout << "hn time3 = ";
      time3.display();                   //显示结果
      cout << endl;
      return 0;
}
```

实践 14　用 Int 类型重载算术运算符

```
//实践 14. cpp
#include < iostream >
using namespace std;
#include < process.h >              //引入 exit()函数

class Int{
    private:
      int i;
    public:
      Int():i(0)                     //无参数构造函数
      {}
```

```
    Int (int ii):i(ii)              //一个参数构造函数
    {}                              //(从 int 转换到 Int)
    void putInt(){                  //显示 Int
      cout << i;
    }
    void getInt(){                  //从 kbd 读入 Int
      cin >> i;
    }
    operator int(){                 //转换运算符(Int 转为 int)
      return i;
    }
    Int operator+(Int i2){          //加
      return check it(long double(i) + long double(i2));
    }
    Int operator-(Int i2){          //减
      return checkit(long double(i) - long double(i2));
    }
    Int operator *(Int i2){         //乘
      return checkit (long double(i) * long double(i2));
    }
    Int operator/(Int i2){          //除
      return checkit (long double(i)/long double(i2));
    }
    Int checkit (long double answer){  //检查结果
      if (answer >2147483647.0L ‖ answer < -214748364).0L)
        { cout <<"hn 溢出错误hn";
          exit(1);
        }
    return Int(int (answer));
    }
};

int main(){
    Int alpha =20;
    Int beta =7;
    Int delta,gamma;
    gamma = alpha + beta;     //27
    cout <<"hn gamma =";
    gamma.putInt();
```

```
    gamma = alpha - beta;     //13
    cout << "\n gamma = ";
    gamma.putInt();
    gamma = alpha * beta;     //140
    cout << "\n gamma = ";
    gamma.putInt();
    gamma = alpha/beta;       //2
    cout << "\n gamma = ";
    gamma.putInt();
    delta = 2147483647;
    gamma = delta + alpha;    //溢出错误
    delta = -2147483647;
    gamma = delta - alpha;    //溢出错误
    cout << endl;
    return 0;
}
```

实践 15　继承 String 类

```
//实践 15.cpp
#include <iostream>
#include <cstring>                          //引入 strcpy()函数等
using namespace std;

class String {                              //基类(父类)
    protected:                              //注意:不可以为 private
        enum {SZ = 80};                     //所有 String 对象的长度
        char str[SZ];                       //保留一个 C 串
    public:
        String(){                           //无参构造函数
          str[0] = '\0';
        }
        String (char s[]){                  //一个参数构造函数
          strcpy(str,s);                    //把串转换成 String
        }
        void display() const {              //显示该 String
          cout << str;
        }
        operator char * (){                 //转换函数
```

```
        return str;                          //把 String 转换成 C 串
      }
};
class Pstring:public String {                //导出类(派生)
    public:
      Pstring (char s[]);                    //构造函数
};
Pstring::Pstring(char s[]){                  //Pstring 的构造函数
    if(strlen(s) > SZ-1){                    //如果太长
      for(int j=0;j<SZ-1;j++)                //首先手工复制
        str[j] = s[j];                       //SZ-1 字符
        str[j] = '\0';                       //添加 null 字符
      }
    else                                     //不太长的话,
      String(s);                             //正常地构造
}

int main(){                                  //定义 String
    Pstring s1 = "This is a very long string which is probably"
                 "no,certainly--going to exceed the limit set by SZ.";
    cout << "\n s1 = ";
    s1.display();                            //显示 String
    Pstring s2 = "This is a short string.";  //定义 String
    cout << "\n s2 = ";
    s2.display();                            //显示 String
    cout << endl;
    return 0;
}
```

实践 16　寻找由用户提供的多个数值的平均值

```
//实践 16.cpp
#include <iostream>
using namespace std;

int main(){
    float flarr[100];                        //数的数组
    char ch;                                 //用户的判定
    int num=0;                               //数的输出计数
```

```
    do {
        cout <<"用户键入数:";              //由用户来键入数
        cin >> * (flarr + num ++);         //直到用户回答'n'为止
        cout <<"是否继续键入(y/n)?";
        cin >>ch;
    }while (ch != 'n');
    float total =0.0;                      //起始 total 为零
    for(int k =0;k <num;k ++)              //把数加到 total 中
        total += * (flarr + k);
    float average =total/m;                //寻找并显示平均值
    cout <<"平均值是" <<average <<endl;
    return 0;
  }
```

实践 17　将指针数组的内容排序到串

```
//实践 17. cpp
#include <iostream>
#include <cstring>                         //引入 strcmp()函数等
const int DAYS =7;                         //数组的指针数

int main(){
    void bsort (char * *,int);             //原型函数
    char* arrptrs [DAYS] =                 //指针数组到字符
        {"Sunday","Monday","Tuesday","Wednesday","Thursday","Friday",
         "Saturday"};
    cout <<"hn Unsorted:hn";
    for (int j =0;j <DAYS;j ++)            //显示未被排序的串
        cout << * (arrptrs + j) <<endl;
    bsort(arrptrs,DAYS);                   //排序的串
    cout <<"hnSorted:hn";
    for(j =0;j <DAYS;j ++)                 //显示已排序的串
        cout << * (arrptrs + j) <<endl;
    return 0;
}
void bsort(char **pp,int n){排序指针到串
    void order (char **,char **);          //原型
    int j,k;                               //索引到数组
    for (j =0;j <n -1;j ++)
```

```
        for (k=j+1;k<n;k++)
    order(pp+j,pp+k);                    //排序指针内容
}
void order(char** pp1,char** pp2){       //两个指针排序
    if(strcmp(*pp1,*pp2)>0){             //如果在第一个串中指针大于第二个
        char *tempptr= *pp1;             //交换这两个指针
        *pp1=*pp2;
        *pp2=tempptr;
    }
}
```

实践 18　创建数组类

```
//实践18.cpp
#include <iostream>
using namespace std;

class Array {//模拟一个正规的C++数组
    private:
        int *ptr;                        //指向Array内容的指针
        int size;                        //Array的长度
    public:
        Array(int s){                    //一个参数的构造函数
          size=s;                        //自变量是Array的长度
          ptr=new int [s];               //为Array申请空间
    }
    ~Array() {                           //析构函数
          delete [] ptr;
    }
    int & operator [] (int j) {          //重载下标操作符
          return *(ptr+j);
    }
};

int main(){
      const int ASIZE=10;                //数组长度
      Array arr(ASIZE);                  //构造一个数组
      for (int j=0;j<ASIZE;j++)          //用方块填充它
          arr [j]=j*j;
```

```
    for (j=0;j<ASIZE;j++)                    //显示其内容
        cout<<arr[j]<<'';
    cout<<endl;
    return 0;
}
```

实践 19　重载赋值运算符和拷贝构造函数

```
//实践 19.cpp
#include <iostream>
using namespace std;

class Array {
  private:
    int * ptr;                               //指向“数组”内容的指针
    int size;                                //数组的长度
  public:
    Array():ptr(0),size(0)                   //无参的构造函数
    {}
    Array (int s):size(s)                    //一个参数的构造函数
    {ptr=new int[s];}
  Array(Array &);                            //拷贝构造函数
  ~Array()                                   //析构函数
      { delete []ptr;}
  int & operator [] (int j)                  //重载下标 op
      { return *(ptr+j);}
  Array & operator=(Array &);                //重载 = 运算符
};
Array::Array(Array & a){                     //拷贝构造函数
    size=a.size;                             //新的一个与老的有相同的长度
    ptr=new int[size];                       //为新的一个内容申请空间
    for(int j=0;j<size;j++)                  //将内容拷贝到新的一个中
        *(ptr+j)=*(a.ptr+j);
}
Array & Array::operator=(Array & a){         //重载 = 运算符
    delete [] ptr;                           //删去老的内容
    size=a.size;                             //使这个对象有相同的长度
    ptr=new int [a.size];                    //获得新的内容的空间
    for (int j=0;j<a.size;j++)               //把内容拷贝到这个对象中
```

```
            *(ptr+j) = *(a.ptr+j);
        return *this;                          //返回这个对象
}

int main(){
        const int ASIZE=10;                    //数组的长度
        Array arr1(ASIZE);                     //创建一个数组
        for (int j=0;j<ASIZE; j++)             //用方块填充
            arr1 [j] = j*j;
        Array arr2(arr1);                      //使用拷贝构造函数
        cout <<"\n arr2:"
        for (j=0;j<ASIZE;j++)                  //检查工作状况
            cout<<arr2 [j]<<"  ";
            Array arr3,arr4;                   //创建两个空 Array 对象
            arr4=arr3=arr1;                    //使用赋值运算符
        cout <<"\n arr3:"
        for (j=0;j<ASIZE:j++)                  //检查 arr3 的工作状况
            cout<<arr3 [j]<<"  ";
        cout<<"\n arr4:";
        for (j=0;j<ASIZE;j++)                  //检查 arr4 的工作状况
            cout<<arr4 [j]<<"  ";
        cout<<endl;
        return 0;
}
```

实践 20　向数组中写数据

```
//实践 20.cpp
#include <iostream>
#include <fstream>                             //引入所有文件流
using namespace std;

class Range {                                  //英制距离类
    private:
        int feet;
        float inches;
    public:
        Range():feet(0),inches(0.0)            //无参构造函数
          {}
```

```
        Range (int ft, float in):feet(ft),inches(in)
          {}                                    //两个参数构造函数
        void getdist(){                         //从用户获得长度
          cout << "\n Enter feet:";
          cin >> feet;
          cout << "Enter inches:";
          cin >> inches;
        }
        void showrange(){                       //显示距离
          cout << feet << "'-" << inches << '"';
        }

int main(){
      char ch;
      Range rang;                               //创建一个 Range 对象
      fstream file;                             //创建输入/输出文件
      file.open("RANG.DAT",ios::binary|ios::app|
                           ios::out|ios::in);   //为添加数据而打开文件
     do {                                       //从用户获得数据写到文件中
          cout << "\n Range";
          rang.getrange();                      //获得一距离
          file.write((char*)& rang, sizeof(rang));  //写到文件中
          cout << "Enter another range (y/n)?";
          cin >> ch;
      } while (ch == 'y');                      //当为'n'时退出
      file.seekg(0);  //重置到文件后始位置
      file.read((char*)& rang, sizeof(rang));  //读第一次距离
      int count = 0;
      while(!file.eof()){  //到 EOF(文件结尾)时退出
          cout << "\n Range" <<++ count << ":";  //显示距离
          rang.showrange();
      file.read((char*)& rang, sizeof(rang));  //读另一个距离
  }
  cout << endl;
  return 0;
}
```

实践 21　模仿 COPY 命令

//实践 21. cpp

```
#include <fstream>                    //引入文件函数
#include <iostream>
using namespace std;
#include <process.h>                 //引入 exit()函数

int main (int argc, char * argv []){
      if (argc !=3){
        cerr <<"\n Format:ocopy srcfile destfile";
        exit(-1);
      }
      char ch;                       //读字符
      ifstream infile;               //创建输入文件
      infile.open(argv[1]);          //打开文件
      if(! infile){                  //检查错误
        cerr <<"\n Can't open" <<argv[1];
        exit(-1);
      }
      ofstream outfile;              //创建输出文件
      outfile.open(argv[2]);         //打开文件
      if(!outfile){                  //检查错误
        cerr <<"\n Can't open " <<argv [2];
        exit(-1);
      }
      while (infile){                //直到遇见 EOF 标志结束
        infile.get(ch);              //读一个字符
        outfile.put(ch);             //写读字符
      }
      return 0;
}
```

实践 22　显示文件的长度

```
//实践 22. cpp
#include <fstream>                   //引入所有文件函数
#include <iostream>
using namespace std;
#include <process.h>                 //引入 exit()函数

int main (int argc, char * argv []){
```

```
    if (argc != 2){
      cerr << "\n Format: filename\n";
      exit(-1);
    }
    ifstream infile;          //创建输入文件
    infile.open(argv[1]);     //打开文件
    if(!infile){              //检查错误
      cerr << "\n Can't open" << argv[1];
      exit(-1);
    }
    infile.seekg(0,ios::end);  //直至文件结尾
    cout << "Size of" << argv[1] << "is" << infile.tellg();//报告字节数
    cout << endl;
    return 0;
}
```

实践 23　使用平均值数组的函数模板

```
//实践 23.cpp
#include <iostream>
using namespace std;
template <class atype>        //函数模板
atype avge(atype * array, int size){
    atype total = 0;
    for (int j = 0;j < size;j ++)     //平均该数组
        total += array [j]
  return (atype)total/size;
}
int intArray []         = {1,3,5,9,11,13};
long longArray []       = {1,3,5,9,11,13};
double doubleArray []   = {1.0,3.0,5.0,9.0,11.0,13.0};
char charArray []       = {1,3,5,9,11,13};

int main(){
    cout << "\n avge(intArray) = " << avge(intArray,6);
    cout << "\n avge(longArray) = " << avge(longArray,6);
    cout << "\n avge(doubleArray) = " << avge(doubleArray,6);
    cout << "\n avge(charArray) = " << avge(charArray,6) << endl;
return 0;
}
```

实践 24　实现作为一模板的队列类，并在处理队列错误中使用例外机制

```
//实践 24.cpp
#include <iostream>
using namespace std;
const int MAX=3;
//- - - - - - - - - - - - - - -
template <class type>
class Queue{
    private:
        Type qu[MAX];                    //任意类型的数组
        int head;                        //队列前端的索引(删去旧的项)
        int tail;                        //队列后端的索引(插入新的项)
        int count;                       //队列中的项数
    public:
        class full{};                    //例外类
        class empty{};
//- - - - - - - - - - - - - - -
    Queue()                              //构造函数(无参)
      { head=-1;tail=-1;count=0;}
    void put(Type var){                  //在队列尾插入一项
      if (count>=MAX)                    //如果队列已满,抛出例外
        throw full();
      qu[++tail]=var                     //否则存储一项
        ++count;
      if (tail>=MAX-1)                   //如果已过了数组尾则反绕回来
        tail=-1
    }
    Type get(){                          //从队列头删去一项
      if (count<=0)                      //如果队列空,则抛出例外
        throw empty();
      Type temp=qu[++head];              //否则得到一项
       --count;
      if (head>=MAX-1)                   //if 已过了数组结尾则反绕回来
        head=-1;
      return temp;                       //返回一项
    }
```

```
};
//- - - - - - - - - - - - - - - -
int main(){
    Queue<float> q1;          //q1 是类 Queue<float>的对象
    float data;               //从用户获得数据项
    char choice='p';          //'x','p'或'g'
    do {                      //do 循环(键入'x'退出)
        try{                  //try 块
          cout<<"\n Enter'x'to exit, 'p' for put, 'g' for get:";
          cin>>choice;
    if(choice=='p'){
        cout<<"Enter data value:";
        cin>>data;
        q1.put(data);
    }
    if(choice=='g')
        cout<<"Data="<<q1.get()<<endl;
    }      //try 块结束
    catch (Queue<float>::full){
        cout<<"Error:queue is full."<<endl;
    }
    catch (Queue<float>::empty){
        cout<<"Error:queue is empty."<<endl;
    }
}while (choice!='x');
return 0;
} //main()结束
```

实践 25　在数组中存储浮点类型数并用 sort()函数排序

```
//实践 25.cpp
#include <iostream>
#include <algorithm>
using namespace std;

int main(){
    int j=0,k;
    char ch;
    float fpn,farr[100];
```

```
    do{
      cout << "Enter a floating point number:";
      cin >> fpn;
      farr [j ++] = fpn;
      cout << "Enter another('y' or 'n')?:";
      cin >> ch;
    }while (ch == 'y');
    sort(farr,farr + j);
    for(k = 0,k < j;k ++)
        cout << farr [k] << ",";
    cout << endl;
    return 0;
}
```

实践 26　使用串对象和 push _ back () 函数的向量

```
//实践 26. cpp
#include < iostream >
#include < string >
#pragma warning (disable:4786)    //仅对 Microsoft
#include < vector >
#include < algorithm >
using namespace std;

int main(){
    vector < string > vectStrings;
    string word;
    char ch;
    do {
        cout << "Enter a word:";
        cin >> word;
        vectStrings.push _ back(word);
        cout << "Enter another ('y' or 'n')?:";
        cin >> ch;
    }while (ch == 'y');
    sort(vectStrings.begin(),vectStrings.end());
    for(int k = 0;k < vectStrings.size();k ++)
        cout << vectStrings [k] << endl;
    return 0;
}
```