第 2 周复习

本周的复习程序将前 14 章介绍的很多技巧组合在一起，编写了一个功能强大的程序。

这个演示用链表使用了虚函数、纯虚函数、函数覆盖、多态、公有继承、函数重载、指针、引用等。

第 13 章提到过链表，另外附录 E 提供了一个健壮的链表使用范例。如果读者还未阅读附录 E 也不用担心，编写链表时使用的 C++代码都是你学过的。注意，这个链表与附录 E 中的链表不同：在 C++中，完成同一项任务的方式有很多。

程序清单 R2.1 旨在创建一个链表。链表的节点用于存储零件，就像工厂中可能用到的一样。虽然这不是该程序的最终版本，但它确实很好地演示一种相当高级的数据结构。该程序清单总长 298 行。阅读输出后面的分析前，请尝试自己分析一下这些代码。

程序清单 R2.1　第 2 周复习程序清单

```
 0: // ***************************************************
 1: //
 2: // Title:      Week 2 in Review
 3: //
 4: // File:       Week2
 5: //
 6: // Description:  Provide a linked list demonstration program
 7: //
 8: // Classes:     PART - holds part numbers and potentially other
 9: //                  information about parts
10: //
11: //              PartNode - acts as a node in a PartsList
12: //
13: //              PartsList - provides the mechanisms for
14: //                          a linked list of parts
15: //
16: //
17: // ***************************************************
18:
19:  #include <iostream>
20:  using namespace std;
21:
22:
23:
24:  // ********** Part **********
25:
26:  // Abstract base class of parts
27:  class Part
28:  {
```

```
29:    public:
30:      Part():itsPartNumber(1) {}
31:      Part(int PartNumber):itsPartNumber(PartNumber){}
32:      virtual ~Part(){};
33:      int GetPartNumber() const { return itsPartNumber; }
34:      virtual void Display() const =0;  // must be overridden
35:    private:
36:      int itsPartNumber;
37:  };
38:
39:  // implementation of pure virtual function so that
40:  // derived classes can chain up

CH12
41:  void Part::Display() const
42:  {
43:     cout << "\nPart Number: " << itsPartNumber << endl;
44:  }
45:
46:  // ********** Car Part ***********
47:
CH12
48:  class CarPart : public Part
49:  {
50:    public:
51:      CarPart():itsModelYear(94){}
52:      CarPart(int year, int partNumber);

CH14
53:      virtual void Display() const
54:      {

CH12
55:         Part::Display(); cout << "Model Year: ";
56:         cout << itsModelYear << endl;
57:      }
58:    private:
59:      int itsModelYear;
60:  };
61:

CH12
62:  CarPart::CarPart(int year, int partNumber):
63:     itsModelYear(year),
64:     Part(partNumber)
65:  {}
66:
67:
68:  // ********** AirPlane Part ***********
69:

CH12
```

```
 70:  class AirPlanePart : public Part
 71:  {
 72:    public:

CH12
 73:      AirPlanePart():itsEngineNumber(1){};
 74:      AirPlanePart(int EngineNumber, int PartNumber);

CH14
 75:      virtual void Display() const
 76:      {

CH12
 77:        Part::Display(); cout << "Engine No.: ";
 78:        cout << itsEngineNumber << endl;
 79:      }
 80:    private:
 81:      int itsEngineNumber;
 82:  };
 83:

CH12
 84:  AirPlanePart::AirPlanePart(int EngineNumber, int PartNumber):
 85:    itsEngineNumber(EngineNumber),
 86:    Part(PartNumber)
 87:  {}
 88:
 89:  // *************** Part Node ************
 90:  class PartNode
 91:  {
 92:    public:
 93:      PartNode (Part*);
 94:      ~PartNode();

CH8
 95:      void SetNext(PartNode * node) { itsNext = node; }

CH8
 96:      PartNode * GetNext() const;
 97:      Part * GetPart() const;
 98:    private:

CH8
 99:      Part *itsPart;
100:      PartNode * itsNext;
101:    };
102:
103:    // PartNode Implementations...
104:
105:    PartNode::PartNode(Part* pPart):
106:    itsPart(pPart),
107:    itsNext(0)
```

```
108:  {}
109:
110:  PartNode::~PartNode()
111:  {
112:    delete itsPart;
113:    itsPart = 0;
114:    delete itsNext;
115:    itsNext = 0;
116:  }
117:
118:  // Returns NULL if no next PartNode
CH8
119:  PartNode * PartNode::GetNext() const
120:  {
121:    return itsNext;
122:  }
123:
124:  Part * PartNode::GetPart() const
125:  {
126:    if (itsPart)
127:      return itsPart;
128:    else
129:      return NULL; //error
130:  }
131:
132:  // ********** Part List ***********
133:  class PartsList
134:  {
135:   public:
136:     PartsList();
137:     ~PartsList();
138:     // needs copy constructor and operator equals!

CH9
139:     Part*     Find(int & position, int PartNumber)  const;
140:     int       GetCount() const { return itsCount; }
141:     Part*     GetFirst() const;

CH9
142:     void      Insert(Part *);
143:     void      Iterate() const;

CH10
144:     Part*     operator[](int) const;
145:   private:

CH8
146:     PartNode * pHead;
147:     int itsCount;
148:  };
149:
150:  // Implementations for Lists...
```

```
151:
152:   PartsList::PartsList():
153:     pHead(0),
154:     itsCount(0)
155:     {}
156:
157:   PartsList::~PartsList()
158:   {
CH8
159:     delete pHead;
160:   }
161:

CH6
162:   Part*  PartsList::GetFirst() const
163:   {
164:     if (pHead)
165:       return pHead->GetPart();
166:     else
167:       return NULL;  // error catch here
168:   }
169:

CH10
170:   Part * PartsList::operator[](int offSet) const
171:   {

CH8
172:     PartNode* pNode = pHead;
173:
174:     if (!pHead)
175:       return NULL; // error catch here
176:
177:     if (offSet > itsCount)
178:       return NULL; // error
179:
180:     for (int i=0;i<offSet; i++)
181:       pNode = pNode->GetNext();
182:
183:    return  pNode->GetPart();
184:   }
185:

CH9
186:   Part*  PartsList::Find(int & position, int PartNumber)  const
187:   {

CH8
188:     PartNode * pNode = 0;
189:     for (pNode = pHead, position = 0;
190:          pNode!=NULL;
191:          pNode = pNode->GetNext(), position++)
```

```
192:     {
193:       if (pNode->GetPart()->GetPartNumber() == PartNumber)
194:         break;
195:     }
196:     if (pNode == NULL)
197:       return NULL;
198:     else
199:       return pNode->GetPart();
200:   }
201:
202:   void PartsList::Iterate() const
203:   {
204:     if (!pHead)
205:       return;

CH8
206:     PartNode* pNode = pHead;
207:     do

CH8
208:       pNode->GetPart()->Display();
209:     while (pNode = pNode->GetNext());
210:   }
211:
212:   void PartsList::Insert(Part* pPart)
213:   {

CH8
214:     PartNode * pNode = new PartNode(pPart);
215:     PartNode * pCurrent = pHead;
216:     PartNode * pNext = 0;
217:
218:     int New = pPart->GetPartNumber();
219:     int Next = 0;
220:     itsCount++;
221:
222:     if (!pHead)
223:     {
224:       pHead = pNode;
225:       return;
226:     }
227:
228:     // if this one is smaller than head
229:     // this one is the new head
230:     if (pHead->GetPart()->GetPartNumber() > New)
231:     {
232:       pNode->SetNext(pHead);
233:       pHead = pNode;
234:       return;
235:     }
236:
237:     for (;;)
```

```
238:     {
239:       // if there is no next, append this new one
240:       if (!pCurrent->GetNext())
241:       {
242:         pCurrent->SetNext(pNode);
243:         return;
244:       }
245:
246:       // if this goes after this one and before the next
247:       // then insert it here, otherwise get the next
248:       pNext = pCurrent->GetNext();
249:       Next = pNext->GetPart()->GetPartNumber();
250:       if (Next > New)
251:       {
252:         pCurrent->SetNext(pNode);
253:         pNode->SetNext(pNext);
254:         return;
255:       }
256:       pCurrent = pNext;
257:     }
258:   }
259:
260:   int main()
261:   {
262:
263:     PartsList pl;
264:

CH8
265:     Part * pPart = 0;
266:     int PartNumber;
267:     int value;
268:     int choice = 99;
269:
270:     while (choice != 0)
271:     {
272:       cout << "(0)Quit (1)Car (2)Plane: ";
273:       cin >> choice;
274:
275:       if (choice != 0 )
276:       {
277:          cout << "New PartNumber?: ";
278:          cin >>  PartNumber;
279:
280:          if (choice == 1)
281:          {
282:            cout << "Model Year?: ";
283:            cin >> value;
284:            pPart = new CarPart(value,PartNumber);
285:          }
286:          else
287:          {
```

```
288:              cout << "Engine Number?: ";
289:              cin >> value;
290:              pPart = new AirPlanePart(value,PartNumber);
291:            }
292:
293:            pl.Insert(pPart);
294:          }
295:        }
296:        pl.Iterate();
297:        return 0;
298:    }
```

输出：

```
(0)Quit (1)Car (2)Plane: 1
New PartNumber?: 2837
Model Year? 90
(0)Quit (1)Car (2)Plane: 2
New PartNumber?: 378
Engine Number?: 4938
(0)Quit (1)Car (2)Plane: 1
New PartNumber?: 4499
Model Year? 94
(0)Quit (1)Car (2)Plane: 1
New PartNumber?: 3000
Model Year? 93
(0)Quit (1)Car (2)Plane: 0

Part Number: 378
Engine No.: 4938

Part Number: 2837
Model Year: 90

Part Number: 3000
Model Year: 93

Part Number: 4499
Model Year: 94
```

分析：

该程序清单提供了一个 Part 对象的链表实现。链表是一种动态数据结构，也就是说它类似于数组，但大小可随对象的添加或删除而改变。链表还包含指向对象的指针，以便将对象链接起来。

这个链表被设计用来存储 Part 类的对象，而 Part 是一个抽象类，用作任何有零件号的类的基类。在这个例子中，从 Part 派生出了子类 CarPart 和 AirplanePart。

Part 类是在第 27～37 行声明的，它由零件号和一些存取器函数组成。假设可以扩展这个类以存储有关零件的其他重要信息，如用在什么部件中、有多少库存等。Part 是一个抽象数据类型，这是由纯虚函数 Display( ) 决定的。

Display( )确实有实现（第 41～44 行）。设计者这样做旨在强制派生类创建自己的 Display( )方法，同时可以调用该方法。

第 48～60 行和第 70～82 行分别声明了两个简单的派生类 CarPart 和 AirPlanePart。每个类都覆盖了 Display( )方法，同时调用了基类的 Display( )方法。

第 90～101 行声明的 PartNode 类用作 Part 类和 PartsList 类之间的接口。它包含一个 Part 指针和一个指向链表中下一节点的指针。PartNode 只包含这样的方法：获取和设置链表中的下一个节点以及返回它指向的 Part。

链表的核心为 PartsList 类，其声明位于第 133～148 行。PartsList 包含一个指向链表中第一个节点的指针（pHead），并使用这个指针通过遍历链表来访问其他所有节点。遍历链表指的是向链表中每个节点询问下一个节点，直到到达 itsNext 指针为 NULL 的节点。

这里只提供部分实现；功能完整的链表将提供对其第一个节点和最后一个节点的更强大访问功能，或提供一个迭代器对象，让客户能够轻松地遍历链表。

然而，PartsList 提供了一些有趣的方法，这些方法按字母顺序排列。这通常是一个不错的注意，因为这使查找函数更容易。

Find( )接受一个 PartNumber 参数和一个 int 参数（第 186～200 行）。如果找到了与 PartNumber 对应的零件，它将返回一个 Part 指针，并将该零件在链表中的位置赋给 int 变量。如果没有找到，则返回 NULL，此时的位置将没有意义。

GetCount( )函数返回链表中节点的数目（第 140 行）。PartsList 将节点数存储在成员变量 itsCount 中，当然也可以通过遍历链表来计算出节点数。

GetFirst( )返回指向链表中第一个 Part 的指针；如果链表为空，则返回 NULL（第 162～168 行）。

在第 212～258 行，Insert( )方法接受一个 Part 指针作为参数，它为该指针指向的零件创建一个 PartNode，然后根据 PartNumber 指定的顺序将 PartNode 插入到链表中。

第 202～210 行的 Iterate( )接受一个指向 Part 的成员函数的指针，后者不接受任何参数，返回类型为 void 且为 const。Iterate( )对链表中的每个 Part 对象调用该函数。在这个范例程序中调用的是 Display( )，这是一个虚函数，因此将根据 Part 对象的运行阶段类型调用正确的 Display( )方法。

在第 170～184 行，重载了下标运算符，Operator[ ]使得可以根据指定的偏移量直接访问 Part。这里还提供了基本的边界检查；如果链表为空或指定的偏移量超过了链表长度，将返回 NULL 作为错误条件。

注意，在实际程序中，这些有关函数的注释放在类声明中。

main( )函数从第 260 开始。第 263 行创建了 PartList 对象。

在第 277 行，重复提示用户选择输入汽车零件还是飞机零件。根据用户的选择，要求用户输入相应的值，并创建正确的零件。创建零件后，第 293 行将其插入到链表中。

PartsList 的 Insert( )方法的实现位于第 212～258 行。输入第一个零件号（2837）后，创建一个零件号为输入值、型号年份为 90 的 CarPart，并将其作为参数传给 LinkedList::Insert( )。

在第 214 行，使用这个零件创建一个新的 PartNode，并用零件号初始化变量 New。第 220 行将 PartsList 的成员变量 itsCount 加 1。

在第 222 行，对 pHead 是否为 NULL 的测试结果为 true。由于这是第一个节点，因此 PartsList 的 pHead 指针为 NULL。因此第 224 行将 pHead 指向新节点，然后函数返回。

接下来用户被提示输入第二个零件，这次输入的是零件号为 378、发动机号为 4938 的 AirPlane 零件。再次调用 PartsList::Insert( )，并用新节点对 pNode 进行初始化。静态成员变量 itsCount 增加到 2，并再次检测 pHead。由于前一次给 pHead 赋了值，因此它不再为 NULL，测试失败。

第 230 行将 pHead 指向的零件的编号 2837 与当前零件号 378 进行比较。由于新零件号小于 pHead 指向的零件的编号，因此第 230 行的测试结果为 true，将 pHead 指向新节点。

第 232 行将新节点设置为指向 pHead 当前指向的节点。注意，这并不是让新节点指向 pHead，而是指向 pHead 当前指向的节点！第 233 行让 pHead 指向新节点。

在第二次循环中，用户输入零件号为 4499、年份为 94 的 CarPart。计数器增加 1，同时这次输入的零件号比 pHead 指向的零件号大，因此进入从第 237 行开始的 for 循环。

pHead 指向的零件的编号为 378；第二个节点中的零件的编号为 2837；当前零件编号为 4499。指针 pCurrent 指向 pHead 指向的节点，因此 itsNext 指针不为 NULL；pCurrent 指向第二个节点，因此第 240 行的测试失败。

指针 pCurrent 被设置为指向下一节点，然后重复循环。这一次第 240 行的测试成功。由于没有下一个节

点，因此第 242 行让当前节点指向新节点，从而完成插入工作。

第四次循环时输入的零件号为 3000。处理过程与前一次循环相同，但这次当前节点指向 2837 时，下一个节点的零件编号为 4499，因此第 250 行的测试返回 TRUE，新节点被插入到这个位置。

当用户最后输入 0 时，第 275 行的测试结果为真，从而结束 while 循环。程序跳到第 296 行执行——调用 Iterate( )，因此跳到第 202 行执行。执行到第 208 行时，使用 Pnode 来访问 Part 并对该 Part 对象调用方法 Display( )。