Osa
VII

23. oppitunti

Mallit

Muutaman viime vuoden aikana on C++ -kieleen lisätty joitakin uusia piirteitä. Eräs kiinnostavimpia ja tehokkaimpia uusia C++ -ominaisuuksia ovat mallit. Mallit mahdollistavat tyyppisuojattujen kokoelmien muodostamisen. Tämän luvun aiheita ovat:

- ❏ Mitä mallit ovat ja kuinka niitä käytetään
- ❏ Miksi mallit tarjoavat paremman vaihtoehdon makroille
- ❏ Kuinka luokkamalleja luodaan

Mitä mallit ovat?

Luvussa 19 sait tietoa linkitetyistä listoista. Tällöin linkitetty lista oli mukavasti kapseloitu: lista tiesi vain alkuosoittimen, alkuosoitin jakoi työnsä sisäisille osoittimille jne,

Eräänä linkitettyjen listojen ongelmana oli se, että ne osaavat käsitellä vain sellaisia tietoja, joiden kanssa ne oli suunniteltu työskentelemään. Jos

Copyright © IT Press – Tämän e-kirjan kopiointi, tulostaminen ja jakeleminen eteenpäin luvatta on kielletty.

halusit sijoittaa jotain muuta linkitettyyn listaasi, et voinut tehdä sitä. Et voinut lisätä listaan (Car-lista tai Cat-lista) jotain solmua, joka oli eri tyyppiä kuin alkuperäisessä listassa.

Ongelman ratkaisemiseksi voidaan luoda List-perusluokka ja johtaa siitä CarList- ja CatsList-luokat. Sitten suuri osa LinkedList-luokasta voidaan upottaa uuteen CatsList-esittelyyn. Myöhemmin haluttaessa luoda lista Car-olioista, muodostetaan uusi luokka ja upotetaan siihen samat LinkedList-rivit kuin edellä.

On tarpeetonta sanoa, että tuo ei ole tyydyttävä ratkaisu. Ennen pitkää joudutaan List-luokkaa ja siitä johdettuja luokkia laajentamaan. Sen jälkeen onkin painajainen varmistaa, että kaikki muutokset siirtyvät kaikkiin vastaaviin luokkiin.

Mallit tarjoavat ratkaisun tähän ongelmaan. Lisäksi ne, toisin kuin vanhanaikaiset makrot, ovat osa kieltä. Ne ovat tyyppisuojattuja ja hyvin joustavia.

Parametrisoidut tyypit

Mallien avulla voidaan opettaa kääntäjää tekemään minkä tahansa tyyppinen lista sen sijaan, että kaikki solmut (jäsenet) olisivat samaa ennalta määritettyä tyyppiä. PartsList on osien lista; CatList on kissalista. Ne eroavat toisistaan ainoastaan listan solmun tyypin mukaan. Mallien avulla listan jäsenen tyypistä tulee parametri luokan määrittelyyn.

Uusi käsite Toiminto, jolla mallista luodaan tietty tyyppi, on ilmentymän luominen ja luotuja luokkia kutsutaan mallin ilmentymiksi.

Uusi käsite Mallit tarjoavat mahdollisuuden luoda yleisluokka ja viedä tuolle luokalle tyyppejä parametreina, jolloin voidaan muodostaa tiettyjä ilmentymiä.

Mallin määrittely

Parametrisoitu List-olio (tai malli listalle) luodaan seuraavasti:

```
1:template <class T>   // esitellään malli ja parametri
2:class List    // parametrisoitava luokka
3:{
4:public:
5:  List();
6:  // luokan esittely
7:};
```

Copyright © IT Press – Tämän e-kirjan kopiointi, tulostaminen ja jakeleminen eteenpäin luvatta on kielletty.

Malliluokan esittelyn ja määrittelyn alussa on varattu sana template. Mallin parametrit ovat tuon avainsanan jälkeen; ne ovat kohteita, jotka muuttuvat jokaisessa ilmentymässä. Esimerkiksi yllä olevassa mallissa muuttuvat listan jäsenten tyypit. Yksi ilmentymä voi tallentaa kokonaislukuja ja toinen taas Animal-olioita.

Tässä esimerkissä käytetään class-avainsanaa, jota seuraa tunniste T. class kertoo, että tämä parametri on tyyppi. Tunnistetta T käytetään kaikkialla muualla mallin määrittelyssä viittaamaan parametrisoituun tyyppiin. Yksi tämän luokan ilmentymä voisi laittaa kokonaisluvun kaikkialle T:n paikalle ja joku muu taas Cat-olion.

Parametrisoidun listaluokan int- ja Cat-ilmentymät esitellään seuraavasti:

```
List<int> anIntList;
List<cat> aCatList;
```

anIntList on kokonaislukulistatyyppiä; aCatList on tyyppiä ListOfCats. Nyt voidaan tyyppiä List<int> käyttää kaikkialla, jossa sitä tavallisesti käytetään: funktion palautusarvona, funktion parametrina jne.

Listaus 23.1 parametrisoi List-oliomme. Tämä on erinomainen menettelytapa mallien muodostamiseen: laita ensin kohteesi toimimaan yksittäisellä tyypillä, kuten luvussa 19, "Linkitetyt listat", tehtiin. Tee sitten parametrisointi, jolla yleistät kohteesi käsittelemään mitä tahansa tyyppiä.

Listaus 23.1. Parametrisoitujen listojen esittely.

```
1:  // **********************************************
2:  //    Tiedosto:        Listaus 23.1
3:  //
4:  //    Tarkoitus:    Esittelee parametrisoidun listan
5:  //    Huomautuksia:
6:  //
7:  //  COPYRIGHT:  Copyright (C) 1997 Liberty Associates, Inc.
8:  //                All Rights Reserved
9:  //
10: // Esittelee oliopohjaisen
11: // näkökulman listan
12: // käsittelyyn.
13: // Käytetän solmuja.
14: // Jaetaan vastuuta.
15: //
16: // Objekti-luokan jäseniä pidetään
17: // listassa.
18: //
19: // **********************************************
20:
21:
22: #include <iostream.h>
23:
24: enum { kIsSmaller, kIsLarger, kIsSame};
25:
26: // Objekti-luokka listaan
27: // Luokkien tuettava seur. metodeita:
28: // Show ja Compare
```

Copyright © IT Press – Tämän e-kirjan kopiointi, tulostaminen ja jakeleminen eteenpäin luvatta on kielletty.

```
29: class Data
30: {
31: public:
32:   Data(int val):myValue(val){}
33:   ~Data()
34: {
35:   cout << "Deleting Data object with value: ";
36:   cout << myValue << "\n";
37: }
38:   int Compare(const Data &);
39:   void Show() { cout << myValue << endl; }
40: private:
41:   int myValue;
42: };
43:
44: // compare päättää
45: // paikan.
46: int Data::Compare(const Data & theOtherObject)
47: {
48:   if (myValue < theOtherObject.myValue)
49:     return kIsSmaller;
50:   if (myValue > theOtherObject.myValue)
51:     return kIsLarger;
52:   else
53:     return kIsSame;
54: }
55:
56: // Toinen luokka listaan.
57: // Tuettava metodeita:
58: // Show ja Compare
59: class Cat
60: {
61: public:
62:   Cat(int age): myAge(age){}
63:   ~Cat()
64:   {
65:     cout << "Deleting ";
66:
67:     out << myAge << " years old Cat.\n";
68:   }
69:   int Compare(const Cat &);
70:   void Show()
71:   {
72:     cout << "This cat is ";
73:     cout << myAge << " years old\n";
74:   }
75: private:
76:   int myAge;
77: };
78:
79:
80: // compare päättää
81: // paikan
82: int Cat::Compare(const Cat & theOtherCat)
83: {
84:   if (myAge < theOtherCat.myAge)
85:     return kIsSmaller;
86:   if (myAge > theOtherCat.myAge)
87:     return kIsLarger;
88:   else
89:     return kIsSame;
```

Copyright © IT Press – Tämän e-kirjan kopiointi, tulostaminen ja jakeleminen eteenpäin luvatta on kielletty.

```
90: }
91:
92:
93: // ADT esittää solmua
94: // Jokaisen johd. luokan korvattava Insert ja Show
95: template <class T>
96: class Node
97: {
98: public:
99:   Node(){}
100:  virtual ~Node(){}
101:  virtual Node * Insert(T * theObject)=0;
102:  virtual void Show() = 0;
103: private:
104: };
105:
106: template <class T>
107: class InternalNode: public Node<T>
108: {
109: public:
110:  InternalNode(T * theObject, Node<T> * next);
111:  ~InternalNode(){ delete myNext; delete myObject; }
112:  virtual Node<T> * Insert(T * theObject);
113:  virtual void Show()
114:  {
115:    myObject->Show();
116:    myNext->Show();
117:  }  // delegoi!
118: private:
119:  T * myObject;  // Itse kohde
120:  Node<T> * myNext;    // Osoittaa seur. solmuun
121: };
122:
123: // Muodostin vain alustaa
124: template <class T>
125: InternalNode<T>::InternalNode(T * theObject, Node<T> * next):
126: myObject(theObject),myNext(next)
127: {
128: }
129:
130: // listan ydin
131: // uusi kohde viedään
132: // solmulle, joka
133: // päättää paikan
134: template <class T>
135: Node<T> * InternalNode<T>::Insert(T * theObject)
136: {
137:
138:  // Onko suurempi vai pienempi?
139:  int result = myObject->Compare(*theObject);
140:
141:
142:  switch(result)
143:  {
144:  // jos sama, edelle
145:  case kIsSame:        // läpi
146:  case kIsLarger:    // ennen
147:  {
148:    InternalNode<T> * ObjectNode =
149:    new InternalNode<T>(theObject, this);
150:    return ObjectNode;
```

Copyright © IT Press – Tämän e-kirjan kopiointi, tulostaminen ja jakeleminen eteenpäin luvatta on kielletty.

```
151:   }
152:
153:   // jos suurempi, seuraavalle,
154:   // joka käsittelee sen.
155:   case kIsSmaller:
156:   myNext = myNext->Insert(theObject);
157:   return this;
158:   }
159:   return this;  // appease MSC
160: }
161:
162:
163: // Häntä
164: template <class T>
165: class TailNode : public Node<T>
166: {
167: public:
168:   TailNode(){}
169:   ~TailNode(){}
170:   virtual Node<T> * Insert(T * theObject);
171:   virtual void Show() { }
172:
173: private:
174:
175: };
176:
177: // Jos mulle, niin ennen
178: // koska olen Häntä
179: template <class T>
180: Node<T> * TailNode<T>::Insert(T * theObject)
181: {
182:   InternalNode<T> * ObjectNode =
183:   new InternalNode<T>(theObject, this);
184:   return ObjectNode;
185: }
186:
187: // Head osoittaa
188: // listan alkuun
189: template <class T>
190: class HeadNode : public Node<T>
191: {
192: public:
193:   HeadNode();
194:   ~HeadNode() { delete myNext; }
195:   virtual Node<T> * Insert(T * theObject);
196:   virtual void Show() { myNext->Show(); }
197: private:
198:   Node<T> * myNext;
199: };
200:
201: // As soon as the head is created
202: // it creates the tail
203: template <class T>
204:
205: HeadNode<T>::HeadNode()
206: {
207:   myNext = new TailNode<T>;
208: }
209:
210: // Mitään ei ennen Headia,
211: // joten seuraavalle.
```

Copyright © IT Press – Tämän e-kirjan kopiointi, tulostaminen ja jakeleminen eteenpäin luvatta on kielletty.

```
212: template <class T>
213: Node<T> * HeadNode<T>::Insert(T * theObject)
214: {
215:   myNext = myNext->Insert(theObject);
216:   return this;
217: }
218:
219: // Saan tulokset
220: template <class T>
221: class LinkedList
222: {
223: public:
224:   LinkedList();
225:   ~LinkedList() { delete myHead; }
226:   void Insert(T * theObject);
227:   void ShowAll() { myHead->Show(); }
228: private:
229:   HeadNode<T> * myHead;
230: };
231:
232: // Aluksi luon Headin
233: // Se luo Tailin
234: // Tyhjä lista siis osoittaa Headiin,
235: // joka osoittaa Tailiin
236: template <class T>
237: LinkedList<T>::LinkedList()
238: {
239:   myHead = new HeadNode<T>;
240: }
241:
242: // Delegoi
243: template <class T>
244: void LinkedList<T>::Insert(T * pObject)
245: {
246:   myHead->Insert(pObject);
247: }
248:
249: // test driver program
250: int main()
251: {
252:   Cat * pCat;
253:   Data * pData;
254:   int val;
255:   LinkedList<Cat>  ListOfCats;
256:   LinkedList<Data> ListOfData;
257:
258:   // Pyydä arvoja
259:   // listaan
260:   for (;;)
261:   {
262:     cout << "What value? (0 to stop): ";
263:     cin >> val;
264:     if (!val)
265:       break;
266:     pCat = new Cat(val);
267:     pData= new Data(val);
268:     ListOfCats.Insert(pCat);
269:     ListOfData.Insert(pData);
270:   }
271:
272: // Läpi listan
```

Copyright © IT Press – Tämän e-kirjan kopiointi, tulostaminen ja jakeleminen eteenpäin luvatta on kielletty.

```
273:   cout << "\n";
274:   ListOfCats.ShowAll();
275:   cout << "\n";
276:   ListOfData.ShowAll();
277:   cout << "\n ************ \n\n";
278:   return 0;  // Lista näkyvistä ja tuhotaan
279: }
```

Tulostus

```
 What value? (0 to stop): 5
 What value? (0 to stop): 13
 What value? (0 to stop): 2
 What value? (0 to stop): 9
 What value? (0 to stop): 7
 What value? (0 to stop): 0

This cat is 2 years old
This cat is 5 years old
This cat is 7 years old
This cat is 9 years old
This cat is 13 years old

2
5
7
9
13

**********************

Deleting Data object with value: 13
Deleting Data object with value: 9
Deleting Data object with value: 7
Deleting Data object with value: 5
Deleting Data object with value: 2
Deleting 13 years old Cat.
Deleting 9 years old Cat.
Deleting 7 years old Cat.
Deleting 5 years old Cat.
Deleting 2 years old Cat.
```

Analyysi

Listaus muistuttaa huomattavasti luvun 19 listausta. Etsi se käsiisi ja huomaat, että muutoksia on vain vähän.

Suurin muutos on siinä, että kukin luokan esittely ja metodi alkaa sanoin

```
template class <T>
```

Koodi kertoo kääntäjälle, että olet parametrisoimassa tätä listaa tyypille, jonka määrittelet myöhemmin tehdessäsi listan ilmentymiä. Esimerkiksi Node-luokan esittely alkaa nyt seuraavasti

Copyright © IT Press – Tämän e-kirjan kopiointi, tulostaminen ja jakeleminen eteenpäin luvatta on kielletty.

```
template <class T>
class node
```

Koodi kertoo, että Node ei esiinny luokkana itsessään, vaan siksi, että aiot luoda ilmentymiä Nodes of Cats ja Nodes of Data. Vietävää, todellista tyyppiä esittää T.

InternalNode on nyt InternalNode<T>. InternalNode<T> ei osoita Data-olioon ja toiseen Node-olioon vaan tunnisteeseen T (mikä tahansa kohdetyyppi) ja kohteeseen Node<T>. Näet tämän riveiltä 104 ja 105.

Kiinnitä huomiosi rivien 135-161 Insert()-metodiin. Logiikka on sama kuin aiemmin, mutta tietyn tyypin sijaan esiintyy koodissa T. Siten rivin 135 parametri on osoitin tunnisteeseen T. Kun myöhemmin luomme tiettyjä listojen ilmentymiä, T korvataan kääntäjän toimesta oikealla tyypillä (Data tai Cat).

Tärkeää on se, että InternalNode voi jatkaa toimimista, erillään todellisesta tyypistä. Se osaa pyytää kohteita vertaamaan itseään. Se ei välitä siitä, tekeekö Cat vertailun samalla tavalla kuin Data. Itse asiassa voimme kirjoittaa tämän siten, että Cat-oliot eivät tallenna ikäänsä; niillä voi olla syntymäaika ja ne voivat laskea suhteellisen ikänsä ajon aikana eikä InternalNode välittäisi siitä mitään.

Mallin kohteiden käyttäminen

Mallin kohteita voidaan käsitellä aivan samoin kuin mitä muita tyyppejä tahansa. Niitä voidaan viedä parametreina, joko arvoina tai viittauksena ja niitä voidaan käyttää funktioiden palautusarvoina, myös arvoina tai viittauksena. Listaus 23.2 osoittaa, kuinka Template-kohteita voidaan viedä funktioon parametreina.

Listaus 23.2. Parametrisoidut listat.

```
1:  // ********************************************
2:  //    Tiedosto:        Listaus 23.2
3:  //
4:  //    Tarkoitus:    Esittelee parametrisoidun listan
5:  //    Huomautuksia:
6:  //
7:  //  COPYRIGHT:  Copyright (C) 1997 Liberty Associates, Inc.
8:  //                All Rights Reserved
9:  //
10: // Esittelee listan
11: // Solmu hoitaa sijoituksen
12: // Solmu on abstrakti
13: // Solmuja ovat head, tail ja sisäinen solmu
14: // Vain sisäinen solmu tallentaa tiedon
15: //
16: // Kohdeluokka on tieto,
17: // joka on listassa.
18: //
```

Copyright © IT Press – Tämän e-kirjan kopiointi, tulostaminen ja jakeleminen eteenpäin luvatta on kielletty.

```
19: // **********************************************
20:
21:
22: #include <iostream.h>
23:
24:
25: enum { kIsSmaller, kIsLarger, kIsSame};
26:
27: // Kohdeluokka
28: //
29: //
30: class Data
31: {
32: public:
33:   Data(int val):myValue(val){}
34:   ~Data()
35:   {
36:     cout << "Deleting Data object with value: ";
37:     cout << myValue << "\n";
38:   }
39:   int Compare(const Data &);
40:   void Show() { cout << myValue << endl; }
41: private:
42:   int myValue;
43: };
44:
45: //
46: //
47:   int Data::Compare(const Data & theOtherObject)
48:   {
49:   if (myValue < theOtherObject.myValue)
50:     return kIsSmaller;
51:   if (myValue > theOtherObject.myValue)
52:     return kIsLarger;
53:   else
54:     return kIsSame;
55:   }
56:
57: //
58: //
59: //
60: class Cat
61: {
62: public:
63:   Cat(int age): myAge(age){}
64:   ~Cat(){cout << "Deleting " << myAge << " years old Cat.\n";}
65:   int Compare(const Cat &);
66:   void Show() { cout << "This cat is " << myAge << " years old\n"; }
67: private:
68:   int myAge;
69: };
70:
71:
72: //
73: //
74: int Cat::Compare(const Cat & theOtherCat)
75: {
76:   if (myAge < theOtherCat.myAge)
77:     return kIsSmaller;
78:   if (myAge > theOtherCat.myAge)
79:     return kIsLarger;
```

Copyright © IT Press – Tämän e-kirjan kopiointi, tulostaminen ja jakeleminen eteenpäin luvatta on kielletty.

```
80:   else
81:     return kIsSame;
82: }
83:
84:
85: //
86: //
87: template <class T>
88: class Node
89: {
90: public:
91:   Node(){}
92:   virtual ~Node(){}
93:   virtual Node * Insert(T * theObject)=0;
94:   virtual void Show() = 0;
95: private:
96: };
97:
98: template <class T>
99: class InternalNode: public Node<T>
100: {
101: public:
102:   InternalNode(T * theObject, Node<T> * next);
103:   ~InternalNode(){ delete myNext; delete myObject; }
104:   virtual Node<T> * Insert(T * theObject);
105:   virtual void Show() // delegoi!
106:   {
107:     myObject->Show(); myNext->Show();
108:   }
109: private:
110:   T * myObject;
111:   Node<T> * myNext;
112: };
113:
114: // Muodostin
115: template <class T>
116: InternalNode<T>::InternalNode(T * theObject, Node<T> * next):
117: myObject(theObject),myNext(next)
118: {
119: }
120:
121: // itse listan ydin
122: //
123: //
124: //
125: template <class T>
126: Node<T> * InternalNode<T>::Insert(T * theObject)
127: {
128:
129:   // vertailu
130:   int result = myObject->Compare(*theObject);
131:
132:
133:   switch(result)
134:   {
135:     // sama ennen
136:     case kIsSame:
137:     case kIsLarger:
138:     {
139:       InternalNode<T> * ObjectNode =
140:       new InternalNode<T>(theObject, this);
```

Copyright © IT Press – Tämän e-kirjan kopiointi, tulostaminen ja jakeleminen eteenpäin luvatta on kielletty.

```
141:       return ObjectNode;
142:     }
143:
144:     // suurempi
145:     // seuraavalle
146:     case kIsSmaller:
147:       myNext = myNext->Insert(theObject);
148:      return this;
149:   }
150:   return this;
151: }
152:
153:
154: // Tail node is just a sentinel
155: template <class T>
156: class TailNode : public Node<T>
157: {
158: public:
159:   TailNode(){}
160:   ~TailNode(){}
161:
162:   virtual Node<T> * Insert(T * theObject);
163:   virtual void Show() { }
164:
165: private:
166:
167: };
168:
169: // Tail
170: // Mitään ei jälkeeni
171: template <class T>
172: Node<T> * TailNode<T>::Insert(T * theObject)
173: {
174:   InternalNode<T> * ObjectNode =
175:   new InternalNode<T>(theObject, this);
176:   return ObjectNode;
177: }
178:
179: // Head
180: // osoittaa alkuun
181: template <class T>
182: class HeadNode : public Node<T>
183: {
184: public:
185:   HeadNode();
186:   ~HeadNode() { delete myNext; }
187:   virtual Node<T> * Insert(T * theObject);
188:   virtual void Show() { myNext->Show(); }
189: private:
190:   Node<T> * myNext;
191: };
192:
193: // Headin jälkeen
194: // tail
195: template <class T>
196: HeadNode<T>::HeadNode()
197: {
198:   myNext = new TailNode<T>;
199: }
200:
201: // Mitään ei ennen Headia
```

Copyright © IT Press – Tämän e-kirjan kopiointi, tulostaminen ja jakeleminen eteenpäin luvatta on kielletty.

```
202: // joten seuraavalle
203: template <class T>
204: Node<T> * HeadNode<T>::Insert(T * theObject)
205: {
206:   myNext = myNext->Insert(theObject);
207:   return this;
208: }
209:
210: // Saan hyödyn
211: template <class T>
212: class LinkedList
213: {
214: public:
215:   LinkedList();
216:   ~LinkedList() { delete myHead; }
217:   void Insert(T * theObject);
218:   void ShowAll() { myHead->Show(); }
219: private:
220:   HeadNode<T> * myHead;
221: };
222:
223: // Aluksi Head
224: // Sitten Tail
225: // Tyhkä lista osoittaa Headiin
226: // Se osoittaa Tailiin
227: template <class T>
228: LinkedList<T>::LinkedList()
229: {
230:   myHead = new HeadNode<T>;
231: }
232:
233: // Delegoi!
234: template <class T>
235: void LinkedList<T>::Insert(T * pObject)
236: {
237:   myHead->Insert(pObject);
238: }
239:
240: void myFunction(LinkedList<Cat>& ListOfCats);
241: void myOtherFunction(LinkedList<Data>& ListOfData);
242:
243: // Testiohjelma
244: int main()
245: {
246:   LinkedList<Cat>  ListOfCats;
247:   LinkedList<Data> ListOfData;
248:
249:   myFunction(ListOfCats);
250:   myOtherFunction(ListOfData);
251:
252:   // Lista läpi
253:   cout << "\n";
254:   ListOfCats.ShowAll();
255:   cout << "\n";
256:   ListOfData.ShowAll();
257:   cout << "\n ************ \n\n";
258:   return 0;  // Tuhotaan!
259: }
260:
261: void myFunction(LinkedList<Cat>& ListOfCats)
262: {
```

Copyright © IT Press – Tämän e-kirjan kopiointi, tulostaminen ja jakeleminen eteenpäin luvatta on kielletty.

```
263:   Cat * pCat;
264:   int val;
265:
266:   // Pyydteään arvoja
267:   // listaan
268:   for (;;)
269:   {
270:     cout << "\nHow old is your cat? (0 to stop): ";
271:     cin >> val;
272:     if (!val)
273:       break;
274:     pCat = new Cat(val);
275:     ListOfCats.Insert(pCat);
276:     }
277: }
278:
279: void myOtherFunction(LinkedList<Data>& ListOfData)
280: {
281:   Data * pData;
282:   int val;
283:
284:   // Arvoja
285:   // listaan
286:
287:   for (;;)
288:   {
289:     cout << "\nWhat value? (0 to stop): ";
290:     cin >> val;
291:     if (!val)
292:       break;
293:     pData = new Data(val);
294:     ListOfData.Insert(pData);
295:     }
296:   }
297: }
```

Tulostus

```
How old is your cat? (0 to stop): 12

How old is your cat? (0 to stop): 2

How old is your cat? (0 to stop): 14

How old is your cat? (0 to stop): 6

How old is your cat? (0 to stop): 0

What value? (0 to stop):  3

What value? (0 to stop):  9

What value? (0 to stop):  1

What value? (0 to stop):  5

What value? (0 to stop):  0
```

Copyright © IT Press – Tämän e-kirjan kopiointi, tulostaminen ja jakeleminen eteenpäin luvatta on kielletty.

```
This cat is 2 years old
This cat is 6 years old
This cat is 12 years old
This cat is 14 years old

1
3
5
9

**************

Deleting Data object with value: 9
Deleting Data object with value: 5
Deleting Data object with value: 3
Deleting Data object with value: 1
Deleting 14 years old Cat.
Deleting 12 years old Cat.
Deleting 6 years old Cat.
Deleting 2 years old Cat.
```

Analyysi
Tämä koodi on samanlaista kuin edellisessä listauksessa, mutta tällä kertaa viedään LinkedLists viittauksena vastaaville funktioille. Kyseessä on tehokas piirre. Kun listat on luotu, niitä voidaan käsitellä täysin määriteltyinä tyyppeinä, viedä funktioille ja palauttaa arvoina.

Standardi mallikirjasto

C++ -kieleen kehitettyjä uusia piirteitä on standardi mallikirjasto (STL, Standard Template Library). Kaikki suuret kääntäjätoimittajat laittavat STL-kirjaston kääntäjänsä mukaan. STL on kirjasto mallipohjaisia luokkia, kuten vektorit, listat, jonot ja pinot. STL sisältää myös joukon yleisalgoritmeja kuten lajittelu ja haku.

STL-kirjaston tavoitteena on tarjota vaihtoehto pyörän uudelleen keksimiselle. STL on testattu ja virheet korjattu, se on hyvin tehokas ja ilmainen! Ja mikä tärkeintä, STL on uudelleen käytettävä; kun osaat käyttää STL-kirjastoa, voit hyödyntää sitä kaikissa ohjelmissasi tarvitsematta vääntää koodia itse.

Yhteenveto

Tässä luvussa opit luomaan ja käyttämään malleja. Mallit ovat osa C++ -kieltä ja niitä käytetään parametrisoitujen tyyppien kehittämiseen. Parametrisoidut tyypit ovat tyyppejä, jotka muuttavat käyttäytymistään sen

Copyright © IT Press – Tämän e-kirjan kopiointi, tulostaminen ja jakeleminen eteenpäin luvatta on kielletty.

mukaan, mitä parametreja on käytetty luomisessa. Niiden avulla voidaan käyttää koodia uudelleen turvallisesti ja tehokkaasti.

Mallin määrittely määrittää parametrisoidun tyypin. Jokainen mallin ilmentymä on todellinen olio, jota voidaan käyttää kuin mitä muuta oliota tahansa - funktion parametrina, palautusarvona, jne.

Kysymyksiä ja Vastauksia

K
Miksi käyttää malleja, kun makrot ovat käytettävissä?
V
Mallit ovat tyyppisuojattuja ja sisältyvät kieleen.

K
Mitä eroa on parametrisoidulla mallifunktiolla ja normaalin funktion parametreilla?
V
Tavallinen funktio ottaa parametreja, joita käytetään toiminnoissa. Mallifunktiolla voidaan parametrisoida tietyn funktioparametrin tyyppi. Voit siis viedä tyyppilistan funktiolle ja antaa sitten mallin ilmentymän määrittää tyypin.

K
Milloin tulisi käyttää malleja ja milloin periyttämistä?
V
Malleja voi käyttää aina, kun kaikki tai lähes kaikki käyttäytyminen pysyy muuttumattomana paitsi sen kohteen tyyppi, jota luokka käyttää. Jos joudut kopioimaan luokan ja muuttamaan yhtä tai useampaa sen jäsentä, saattaisi olla tehokkaampaa käyttää mallia.

Copyright © IT Press – Tämän e-kirjan kopiointi, tulostaminen ja jakeleminen eteenpäin luvatta on kielletty.