Osa VI

Erikoisaiheet

Oppitunnit

Osa
VI

20. oppitunti

Erikoisluokat ja -funktiot

C++ tarjoaa useita keinoja rajoittaa muuttujien ja osoittimien näkyvyysaluetta ja vaikutusta. Toistaiseksi olet nähnyt, kuinka globaalit muuttujat, paikallisten funktioiden muuttujat, osoittimet ja luokan jäsenmuuttujat luodaan. Tässä luvussa käsitellään seuraavia aiheita:

- Mitä ovat staattiset jäsenmuuttujat ja staattiset jäsenfunktiot
- Kuinka staattisia jäsenmuuttujia ja -funktioita käytetään
- Mitä ovat ystäväfunktiot ja ystäväluokat
- Kuinka ystäväfunktioilla ratkaistaan erikoisongelmia

Staattiset jäsenmuuttujat

Tähän asti olet varmaankin pitänyt kunkin olion tietoa ainutkertaisena tuolle oliolle, jolloin tietoa ei ole voitu jakaa muille luokan olioille. Jos sinulla on esimerkiksi viisi Cat-oliota, niillä kullakin on oma ikä-, paino- ja muut tietonsa. Yhden olion ikä ei vaikuta toisen olion ikään.

On kuitenkin tilanteita, joissa käytät tietoa, joka on jaettu usean luokan olion kesken. Saatat esimerkiksi haluta tietää, kuinka monta Cat-oliota on toistaiseksi syntynyt ja kuinka monta on vielä elossa.

Toisin kuin muut jäsenmuuttujat ovat staattiset jäsenmuuttujat jaettavissa usean eri luokan ilmentymän kesken. Ne ovat globaalin tiedon, joka on kaikkien osien käsiteltävissä ja jäsentiedon, joka on tavallisesti kunkin olion käytettävissä, välinen kompromissi.

Voit ajatella, että staattinen jäsentieto kuuluu pikemminkin luokalle kuin yksittäiselle oliolle. Normaalia jäsentietoa on aina yksi oliota kohti, mutta staattisia jäseniä on yksi luokkaa kohti. Listaus 20.1 esittelee Cat-olion, jossa on staattinen tietojäsen, HowManyCats. Tuo muuttuja seuraa luotavien Cat-olioiden määrää. Se toteutetaan kasvattamalla staattista muuttujaa, HowManyCats, luotaessa uusi olio ja vähentämällä kyseistä arvoa, kun olio tuhotaan.

Listaus 20.1. Staattinen jäsentieto.

```
1:  //Listaus 20.1 Staattinen jäsentieto.
2:
3:  #include <iostream.h>
4:
5:  class Cat
6:  {
7:  public:
8:    Cat(int age = 1):itsAge(age){HowManyCats++; }
9:    virtual ~Cat() { HowManyCats--; }
10:   virtual int GetAge() { return itsAge; }
11:   virtual void SetAge(int age) { itsAge = age; }
12:   static int HowManyCats;
13:
14: private:
15:   int itsAge;
16:
17: };
18:
19:   int Cat::HowManyCats = 0;
20:
21: int main()
22: {
23:   const int MaxCats = 5;
24:   Cat *CatHouse[MaxCats], i;
25:   for (i = 0; i<MaxCats; i++)
26:     CatHouse[i] = new Cat(i);
27:
28:     for (i = 0; i<MaxCats; i++)
29:     {
30:       cout << "There are ";
```

```
31:       cout << Cat::HowManyCats;
32:       cout << " cats left!\n";
33:       cout << "Deleting the one which is ";
34:       cout << CatHouse[i]->GetAge();
35:       cout << " years old\n";
36:       delete CatHouse[i];
37:       CatHouse[i] = 0;
38:     }
39:   return 0;
40: }
```

Tulostus

```
There are 5 cats left!
Deleting the one which is 0 years old
There are 4 cats left!
Deleting the one which is 1 years old
There are 3 cats left!
Deleting the one which is 2 years old
There are 2 cats left!
Deleting the one which is 3 years old
There are 1 cats left!
Deleting the one which is 4 years old
```

Analyysi

Riveillä 5-17 esitellään yksinkertaistettu Cat-luokka. Rivillä 12 esitellään HowManyCats staattisena, int-tyyppisenä jäsenmuuttujana.

HowManyCats-muuttujan esittely ei määrittele kokonaislukua; mitään tilaa ei varata. Toisin kuin tavallisten jäsenmuuttujien kohdalla, ei tilaa varata staattisille jäsenmuuttujille uuden Cat-olion luomisen yhteydessä, koska HowManyCats ei ole olion sisällä. Sen takia muuttuja määritellään ja alustetaan rivillä 19.

On yleinen virhe unohtaa määritellä luokkien staattiset jäsenmuuttujat. Varo sitä! Jos näin tapahtuu, linkittäjä heittää sinulle seuraavanlaisen virheilmoituksen:

```
undefined symbol cat::HowManyCats
```

Määrittelyä ei tarvitse tehdä itsAge-muuttujalle, koska se ei ole staattinen muuttuja ja se määritellään aina uusia olioita luotaessa (nyt rivillä 26).

Cat-muodostin kasvattaa staattista jäsenmuuttujaa rivillä 8. Tuhoajafunktio taas vähentää sitä rivillä 9. Siten HowManyCats sisältää aina oikean Cat-olioiden määrän.

Rivien 21-39 pääohjelma luo 5 Cat-oliota ja sijoittaa ne taulukkoon. Tällöin kutsutaan siis muodostinta 5 kertaa ja samalla kasvatetaan HowManyCats-muuttujaa.

Ohjelma käy sitten silmukassa läpi taulukon ja tulostaa HowManyCats-muuttujan arvon ennen kyseisen Cat-olion osoittimen tuhoamista. Tulostus kertoo, että alkuarvo on 5 ja jokaisella silmukkakierroksella arvo vähenee yhdellä.

Huomaa, että HowManyCats on public-tyyppinen ja main() voi käsitellä sitä suoraan. Ei ole kuitenkaan mitään syytä tehdä näin. Olisi parempi tehdä siitä private-tyyppinen muiden jäsenmuuttujien tapaan ja lisätä ohjelmaan käsittelymetodi ainakin, jos tietoa halutaan käsitellä kunkin Cat-ilmentymän kautta. Toisaalta, jos tietoa halutaan käsitellä suoraan ilman Cat-oliota, on tarjolla kaksi vaihtoehtoa: pidä se julkisena tai käytä staattista jäsenfunktiota.

Staattiset jäsenfunktiot

Staattiset jäsenfunktiot ovat staattisten jäsenmuuttujien kaltaisia: ne eivät ole olion sisällä vaan luokan näkyvyysalueella. Siksi niitä voidaan kutsua ilman luokan olioita, kuten listaus 20.2 esittää.

Listaus 20.2. Staattiset jäsenfunktiot.

```
1:  //Listaus 20.2 Staattiset jäsentiedot
2:
3:  #include <iostream.h>
4:
5:  class Cat
6:  {
7:  public:
8:    Cat(int age = 1):itsAge(age){HowManyCats++; }
9:    virtual ~Cat() { HowManyCats--; }
10:   virtual int GetAge() { return itsAge; }
11:   virtual void SetAge(int age) { itsAge = age; }
12:   static int GetHowMany() { return HowManyCats; }
13: private:
14:   int itsAge;
15:   static int HowManyCats;
16: };
17:
18: int Cat::HowManyCats = 0;
19:
20: void TelepathicFunction();
21:
22: int main()
23: {
24:   const int MaxCats = 5;
25:   Cat *CatHouse[MaxCats],i;
26:   for (i = 0; i<MaxCats; i++)
27:   {
28:     CatHouse[i] = new Cat(i);
29:     TelepathicFunction();
30:   }
31:
32:   for ( i = 0; i<MaxCats; i++)
33:   {
34:     delete CatHouse[i];
```

```
35:     TelepathicFunction();
36:   }
37:   return 0;
38: }
39:
40: void TelepathicFunction()
41: {
42:   cout << "There are " << Cat::GetHowMany() << " cats alive!\n";
43: }
```

Tulostus

```
There are 1 cats alive
There are 2 cats alive
There are 3 cats alive
There are 4 cats alive
There are 5 cats alive
There are 4 cats alive
There are 3 cats alive
There are 2 cats alive
There are 1 cats alive
There are 0 cats alive
```

Analyysi

Rivillä 15 esitellään staattinen jäsenmuuttuja HowManyCats private-osassa. Julkinen käsittelyfunktio, GetHowMany(), esitellään sekä public-tyyppisenä että staattisena rivillä 12.

Koska GetHowMany() on julkinen, jokainen funktio voi kutsua sitä ja koska se on staattinen, ei tarvita Cat-oliota sen kutsumiseen. Siksi rivin 40 TelepathicFunction() voi käyttää funktiota ilman Cat-oliota. Tietenkin GetHowMany()-funktiota olisi voitu kutsua main()-funktion Cat-olioista käsin, kuten mitä tahansa muita tavallisia jäsenfunktioita.

Huom! Staattisilla jäsenfunktioilla ei ole this-osoitinta; siksi niitä ei voida esitellä const-tyyppisinä. Ja vielä, koska jäsenmuuttujia käsitellään jäsenfunktioissa this-osoittimen avulla, eivät staattiset jäsenfunktiot voi käsitellä mitään ei-staattisia jäsenmuuttujia.

Koosteluokat

Kuten olet aiemmista esimerkeistä nähnyt, voivat luokan jäsenmuuttujat sisältää toisen luokan olioita. C++ -ohjelmoijat sanovat, että ulompi luokka sisältää sisemmän luokan. Siksi Employee-luokka saattaa sisältää merkkijono-olioita (esimerkiksi työntekijän nimen) sekä kokonaislukuja (esimerkiksi palkka) jne.

Listauksessa 20.3 on suppea, mutta hyödyllinen String-luokka.

Listaus 20.3. String-luokka.

```
1:  #include <iostream.h>
2:  #include <string.h>
3:
4:  class String
5:  {
6:  public:
7:    // muodostimet
8:    String();
9:    String(const char *const);
10:   String(const String &);
11:   ~String();
12:
13:   // ylimääritellyt operaattorit
14:   char & operator[](int offset);
15:   char operator[](int offset) const;
16:   String operator+(const String&);
17:   void operator+=(const String&);
18:   String & operator= (const String &);
19:
20:   // käsittelijät
21:   int GetLen()const { return itsLen; }
22:   const char * GetString() const { return itsString; }
23:   // static int ConstructorCount;
24:
25: private:
26:   String (int);         // private-muodostin
27:   char * itsString;
28:   int itsLen;
29:
30: };
31:
32: // oletusmuodostin luo 0-tavuisen merkkijonon
33: String::String()
34: {
35:   itsString = new char[1];
36:   itsString[0] = '\0';
37:   itsLen=0;
38:   // cout << "\tDefault string constructor\n";
39:   // ConstructorCount++;
40: }
41:
42: // private-tukimuodostin
43: // auttaa metodeita luomaan tietyn kokoisen
44: // merkkijonon. Täytetään nullilla.
45: String::String(int len)
46: {
47:   itsString = new char[len+1];
48:   int i;
49:   for (i = 0; i<=len; i++)
50:   itsString[i] = '\0';
51:   itsLen=len;
52:   // cout << "\tString(int) constructor\n";
53:   // ConstructorCount++;
54: }
55:
56: String::String(const char * const cString)
57: {
58:   itsLen = strlen(cString);
59:   itsString = new char[itsLen+1];
60:   int i;
```

```
61:   for (i = 0; i<itsLen; i++)
62:     itsString[i] = cString[i];
63:   itsString[itsLen]='\0';
64:   // cout << "\tString(char*) constructor\n";
65:   // ConstructorCount++;
66: }
67:
68: String::String (const String & rhs)
69: {
70:   itsLen=rhs.GetLen();
71:   itsString = new char[itsLen+1];
72:   int i;
73:   for ( i = 0; i<itsLen;i++)
74:     itsString[i] = rhs[i];
75:   itsString[itsLen] = '\0';
76:   // cout << "\tString(String&) constructor\n";
77:   // ConstructorCount++;
78: }
79:
80: String::~String ()
81: {
82:   delete [] itsString;
83:   itsLen = 0;
84:   // cout << "\tString destructor\n";
85: }
86:
87: // =-operaattori: vapauttaa muistin
88: // kopioi sitten merkkijonon
89: String& String::operator=(const String & rhs)
90: {
91:   if (this == &rhs)
92:     return *this;
93:   delete [] itsString;
94:   itsLen=rhs.GetLen();
95:   itsString = new char[itsLen+1];
96:   int i;
97:   for (i = 0; i<itsLen;i++)
98:     itsString[i] = rhs[i];
99:   itsString[itsLen] = '\0';
100:  return *this;
101:  // cout << "\tString operator=\n";
102: }
103:
104: //ei-vakio indeksi-operaattori palauttaa
105: // viittauksen merkkiin, joten sitä
106: // voidaan muuttaa!
107: char & String::operator[](int offset)
108: {
109:   if (offset > itsLen)
110:     return itsString[itsLen-1];
111:   else
112:     return itsString[offset];
113: }
114:
115: // vakio indeksioperaattori toimii
116: // vakioilla olioilla (kts kopiomuodostin)
117: char String::operator[](int offset) const
118: {
119:   if (offset > itsLen)
120:     return itsString[itsLen-1];
121:   else
```

```
122:     return itsString[offset];
123: }
124:
125: // luo uuden merkkijonon lisäämällä nykyisen
126: // merkkijonon rhs:ään.
127: String String::operator+(const String& rhs)
128: {
129:   int  totalLen = itsLen + rhs.GetLen();
130:   int i,j;
131:   String temp(totalLen);
132:   for (i = 0; i<itsLen; i++)
133:     temp[i] = itsString[i];
134:   for (j = 0; j<rhs.GetLen(); j++, i++)
135:     temp[i] = rhs[j];
136:   temp[totalLen]='\0';
137:   return temp;
138: }
139:
140: // muuttaa nykyistä merkkijonoa, ei palauta mitään
141: void String::operator+=(const String& rhs)
142: {
143:   int rhsLen = rhs.GetLen();
144:   int totalLen = itsLen + rhsLen;
145:   int i,j;
146:   String  temp(totalLen);
147:   for (i = 0; i<itsLen; i++)
148:     temp[i] = itsString[i];
149:   for (j = 0; j<rhs.GetLen(); j++, i++)
150:     temp[i] = rhs[i-itsLen];
151:   temp[totalLen]='\0';
152:   *this = temp;
153: }
```

Tulostus

Ei tulostusta

Analyysi

Rivillä 23 esitellään staattinen jäsenmuuttuja ConstructorCount, joka sitten alustetaan rivillä 152. Tuota muuttujaa kasvatetaan jokaisen merkkijonomuodostimen yhteydessä.

Listaus 20.4 kuvaa Employee-luokan, joka sisältää 3 merkkijono-oliota. Huomaa, että osa lauseista on kommentoitu ohitettaviksi; niitä käytetään myöhemmissä listauksissa.

Listaus 20.4. Employee-luokka ja pääohjelma.

```
1:  class Employee
2:  {
3:
4:  public:
5:    Employee();
6:    Employee(char *, char *, char *, long);
7:    ~Employee();
8:    Employee(const Employee&);
9:    Employee & operator= (const Employee &);
10:
```

```
11:    const String & GetFirstName() const { return itsFirstName; }
12:    const String & GetLastName() const { return itsLastName; }
13:    const String & GetAddress() const { return itsAddress; }
14:    long GetSalary() const { return itsSalary; }
15:
16:    void SetFirstName(const String & fName) { itsFirstName = fName; }
17:    void SetLastName(const String & lName) { itsLastName = lName; }
18:    void SetAddress(const String & address) { itsAddress = address; }
19:    void SetSalary(long salary) { itsSalary = salary; }
20:  private:
21:    String    itsFirstName;
22:    String    itsLastName;
23:    String    itsAddress;
24:    long      itsSalary;
25:  };
26:
27:  Employee::Employee():
28:  itsFirstName(""),
29:  itsLastName(""),
30:  itsAddress(""),
31:  itsSalary(0)
32:  {}
33:
34:  Employee::Employee(char * firstName, char * lastName,
35:  char * address, long salary):
36:  itsFirstName(firstName),
37:  itsLastName(lastName),
38:  itsAddress(address),
39:  itsSalary(salary)
40:  {}
41:
42:  Employee::Employee(const Employee & rhs):
43:  itsFirstName(rhs.GetFirstName()),
44:  itsLastName(rhs.GetLastName()),
45:  itsAddress(rhs.GetAddress()),
46:  itsSalary(rhs.GetSalary())
47:  {}
48:
49:  Employee::~Employee() {}
50:
51:  Employee & Employee::operator= (const Employee & rhs)
52:  {
53:    if (this == &rhs)
54:      return *this;
55:
56:    itsFirstName = rhs.GetFirstName();
57:    itsLastName = rhs.GetLastName();
58:    itsAddress = rhs.GetAddress();
59:    itsSalary = rhs.GetSalary();
60:
61:    return *this;
62:  }
63:
64:  int main()
65:  {
66:    Employee Edie("Jane","Doe","1461 Shore Parkway", 20000);
67:    Edie.SetSalary(50000);
68:    String LastName("Levine");
69:    Edie.SetLastName(LastName);
70:    Edie.SetFirstName("Edythe");
71:
```

```
72:    cout << "Name: ";
73:    cout << Edie.GetFirstName().GetString();
74:    cout << " " << Edie.GetLastName().GetString();
75:    cout << ".\nAddress: ";
76:    cout << Edie.GetAddress().GetString();
77:    cout << ".\nSalary: " ;
78:    cout << Edie.GetSalary();
79:    return 0;
80: }
```

Tulostus

```
Name: Edythe Levine
Address: 1461 Shore Parkway
Salary: 50000
```

Huom! Laita listauksen 20.3 koodi tiedostoon nimeltä STRING.HPP. Kun sitten tarvitset String-luokkaa, käytä #include-komentoa. Lisää esimerkiksi listauksen 20.4 alkuun lause #include String.hpp. Tällöin String-luokka lisätään ohjelmaasi.

Analyysi

Listauksessa 20.4 on Employee-luokka, joka sisältää 3 merkkijono-oliota: itsFirstname, itsLastname ja itsAddress.

Rivillä 66 luodaan Employee-olio, joka alustetaan neljällä arvolla. Rivillä 67 kutsutaan metodia SetSalary() käyttäen vakioarvoa 50000. Huomaa, että todellisessa ohjelmassa arvo olisi joko dynaaminen (ajonaikana asetettava) tai vakio.

Rivillä 68 luodaan merkkijono ja alustetaan se C++ -merkijonovakiolla. Tuota merkkijono-oliota käytetään sitten SetLastName()-metodin argumenttina rivillä 69.

Rivillä 70 kutsutaan SetFirstName()-metodia, jolle viedään toinen merkkijonovakio. Jos kuitenkin kiinnität enemmän huomiota koodiin, havaitset, että SetFirstName() ei ota merkkijonoa argumentikseen vaan se vaatii vakiomerkkijonoviittauksen.

Kääntäjä ratkaisee tämän, koska se tietää, kuinka merkkijono muodostetaan vakiosta merkkijonosta. Sehän kerrottiin rivillä 9.

Koosteluokan jäsenten käsittely

Employee-olioilla ei ole erikoispääsyä String-jäsenmuuttujiin. Jos Edie-olio yrittäisi käsitellä oman jäsenmuuttujansa itsFirstName jäsenmuuttujaa itsLen, syntyisi kääntämisvirhe. Siitä ei ole kuitenkaan paljoa haittaa. Metodit tarjoavat liittymän String-luokkaan eikä Employee-luokan tarvitse huolehtia yksityiskohtien toteuttamisesta.

Sisältönä olevien jäsenten käsittelyn suodattaminen

Huomaa, että String-luokalla on operator+. Employee-luokan suunnittelija on tukkinut pääsyn operator+ -elementtiin Employee-olioilta määrittämällä, että kaikki merkkijonon käsittelyfunktiot (kuten GetFirstName()) palauttavat vakioviittauksen. Koska operator+ ei ole (eikä voi olla) vakiofunktio (koska se muuttaa kutsuttua oliota), aiheuttaa seuraavan lauseen kirjoittaminen kääntämisvirheen:

```
String buffer = Edie.GetFirstName() + Edie.GetLastName();
```

GetFirstName() palauttaa vakion String-olion eikä operator+:aa voi käyttää vakio-oliolle.

Tilanne korjataan ylikuormittamalla GetFirstName() olemaan ei-vakio:

```
const String & GetFirstname() const { return itsFirstName; }
String & GetFirstName() { return itsFirstName; }
```

Huomaa, että palautusarvo ei enää ole vakio ja että jäsenfunktio itse ei enää ole vakio. Palautusarvon muuttaminen ei riitä funktion nimen ylikuormittamiseen; on muutettava myös funktion itsensä vakiomuotoisuus.

Koostumisen haittapuolet

On tärkeää huomata, että Employee-luokan käyttäjä maksaa hinnan noista merkkijono-olioista joka kerta kun sellainen muodostetaan tai tehdään Employee-kopio.

Poistamalla kommenttimerkit listauksen 20.3 cout-lauseista (rivit 38, 52, 64, 76, 84 ja 101) saadaan paljastettua, kuinka usein muodostimia kutsutaan.

Kopioiminen arvona vai viittauksena

Kun viet Employee-oliot arvoina, kaikki niiden sisältämät merkkijonot kopioidaan myöskin ja siksi kutsutaan kopiomuodostinta. Se on hyvin tuhlailevaa, vie muistia ja aikaa.

Kun viet Employee-oliot viittauksena (osoittimien tai viittausten avulla), vältytään tuhlailulta. Juuri siksi C++ -ohjelmoijat yrittävät olla viemättä arvoina suurempia kuin neljä tavua olevia kohteita.

Ystäväluokat

Joskus halutaan luokkia luoda yhtenä joukkona. Nuo yhdistetyt luokat voivat haluta käsitellä toistensa private-jäseniä, mutta itse haluaisit ehkä tehdä tiedosta julkista.

Jos haluat antaa private-tyyppiset tietosi ja funktiosi muiden luokkien käyttöön, on sinun esiteltävä nuo muut luokat ystävinä (friend). Tällöin laajennat luokkasi liittymää sisältämään ystäväluokan.

On tärkeää huomata, että ystävyyttä ei voida siirtää. Se, että sinä olet minun ystäväni ja Joe on sinun ystäväsi, ei tarkoita, että Joe olisi myös minun ystäväni. Ystävyys ei myöskään periydy. Vaikka olenkin sinun ystäväsi, ei tarkoita, että olen automaattisesti myös lastesi ystävä.

Ystävyys ei ole myöskään kommutatiivinen. Jos määrään Class Onen olemaan Class Twon ystävä, en samalla määrää sitä, että Class Two on Class Onen ystävä. Käänteisyys tai vaihdannaisuus eivät siis ole voimassa.

Ystäväluokkien esittelyissä tulee olla huolellinen. Jos kaksi luokkaa on läheisessä suhteessa toisiinsa ja toisen luokan tulee säännöllisesti käsitellä toisen tietoa, on ystäväluokille ehkä olemassa perusteet. Mutta käytä sitä säästeliäästi; usein on aivan yhtä helppoa käyttää public-metodeita ja tällöin voit käsitellä toista luokkaa kääntämättä uudelleen toista.

Huom! Aloittelevat C++ -ohjelmoijat valittavat usein, että ystävä-määrittelyt vähentävät kapseloinnin merkitystä oliopohjaisessa ohjelmoinnissa. Se on kuitenkin tyhjää puhetta. Ystävä-määrittely tekee ystävästä osan luokan liittymää eikä se huononna kapselointia sen enempää kuin julkinen jakaminenkaan.

Ystävä-funktiot

Toisinaan on tarvetta myöntää ystävätason pääsy koko luokan sijaan vain muutamalle luokan funktiolle. Se toteutetaan esittelemällä toisen luokan jäsenfunktiot ystävinä sen sijaan, että koko luokka olisi ystävä. Itse asiassa voit esitellä minkä tahansa, olipa se toisen luokan jäsenfunktio tai ei, ystäväfunktiona.

Osoittimet funktioihin

Samalla lailla kuin taulukon nimi on vakio-osoitin taulukon ensimmäiseen alkioon, on funktion nimi vakio-osoitin funktioon. On mahdollista esitellä osoitinmuuttuja, joka osoittaa funktioon ja kutsua sitten funktiota osoit-

timen avulla. Tällainen voi olla hyvin hyödyllistä; se sallii luoda ohjelmia, joissa päätetään, mitä funktioita kutsutaan käyttäjän syötön perusteella.

Ainoa hankala kohta funktio-osoittimissa on ymmärtää osoitetun kohteen tyyppi. int-tyyppinen osoitin osoittaa kokonaislukuun ja funktio-osoitin osoittaa funktioon, jolla on sopiva palautustyyppi ja allekirjoitus.

Esittelyssä

```
long (* funcPtr) (int);
```

on funcPtr osoitin (huomaa * nimen edessä), joka osoittaa funktioon, joka ottaa parametrikseen kokonaisluvun ja palauttaa long-arvon. Sulkumerkit merkinnän * funcPtr ympärillä ovat välttämättömät, koska int-sulkumerkit sitovat tiukemmin eli niillä on korkeampi prioriteetti kuin epäsuoruusoperaattorilla (*). Ilman ensimmäisiä sulkumerkkejä esittelisi lause funktion, joka ottaa kokonaisluvun ja palauttaa osoittimen long-tyyppiin. (Huomaa, että välilyönnit ovat tässä merkityksettömiä.)

Tutki seuraavia esittelyjä:

```
long * Function (int);
long (* funcPtr) (int);
```

Ensimmäinen esittely on funktio, joka ottaa kokonaisluvun ja palauttaa osoittimen long-tyyppiseen muuttujaan. Toinen esittely on osoitin funktioon, joka ottaa kokonaisluvun ja palauttaa long-tyyppisen muuttujan.

Funktio-osoittimen esittely sisältää aina palautustyypin ja sulkumerkit, joiden sisällä ovat parametrien tyypit, mikäli niitä on. Listaus 20.5 havainnollistaa funktio-osoittimien esittelyä ja käyttöä.

Listaus 20.5. Osoittimet funktioihin.

```
1:  // Listaus 20.5 Funktio-osoittimet
2:
3:  #include <iostream.h>
4:
5:  void Square (int&,int&);
6:  void Cube (int&, int&);
7:  void Swap (int&, int &);
8:  void GetVals(int&, int&);
9:  void PrintVals(int, int);
10: enum BOOL { FALSE, TRUE };
11:
12: int main()
13: {
14:   void (* pFunc) (int &, int &);
15:   BOOL fQuit = FALSE;
16:
17:   int valOne=1, valTwo=2;
18:   int choice;
19:   while (fQuit == FALSE)
20:   {
```

```
21:     cout << "(0)Quit (1)Change Values (2)Square (3)Cube (4)Swap: ";
22:     cin >> choice;
23:   switch (choice)
24:   {
25:     case 1: pFunc = GetVals; break;
26:     case 2: pFunc = Square; break;
27:     case 3: pFunc = Cube; break;
28:     case 4: pFunc = Swap; break;
29:     default : fQuit = TRUE; break;
30:   }
31:
32:   if (fQuit)
33:     break;
34:
35:   PrintVals(valOne, valTwo);
36:   pFunc(valOne, valTwo);
37:   PrintVals(valOne, valTwo);
38: }
39:   return 0;
40: }
41:
42: void PrintVals(int x, int y)
43: {
44:   cout << "x: " << x << " y: " << y << endl;
45: }
46:
47: void Square (int & rX, int & rY)
48: {
49:   rX *= rX;
50:   rY *= rY;
51: }
52:
53: void Cube (int & rX, int & rY)
54: {
55:   int tmp;
56:
57:   tmp = rX;
58:   rX *= rX;
59:   rX = rX * tmp;
60:
61:   tmp = rY;
62:   rY *= rY;
63:   rY = rY * tmp;
64: }
65:
66: void Swap(int & rX, int & rY)
67: {
68:   int temp;
69:   temp = rX;
70:   rX = rY;
71:   rY = temp;
72: }
73:
74: void GetVals (int & rValOne, int & rValTwo)
75: {
76:   cout << "New value for ValOne: ";
77:   cin >> rValOne;
78:   cout << "New value for ValTwo: ";
79:   cin >> rValTwo;
80: }
```

Tulostus

```
(0)Quit (1)Change Values (2)square (3)Cube (4)Swap: 1
x: 1 y:2
New value for ValOne: 2
New value for ValTwo: 3
x: 2 y:3
(0)Quit (1)Change Values (2)square (3)Cube (4)Swap: 3
x: 2 y:3
x: 8 y:27
(0)Quit (1)Change Values (2)square (3)Cube (4)Swap: 2
x: 8 y:27
x: 64 y:729
(0)Quit (1)Change Values (2)square (3)Cube (4)Swap: 2
x: 64 y:729
x: 729 y:64
(0)Quit (1)Change Values (2)square (3)Cube (4)Swap: 0
```

Analyysi

Riveillä 5-8 esitellään neljä funktiota, joilla on sama palautustyyppi ja allekirjoitus; ne palauttavat void-arvon ja ottavat parametreikseen kaksi viittausta kokonaislukuihin.

Rivillä 14 esitellään pFunc osoittimena funktioon, joka palauttaa void-arvon ja ottaa kaksi kokonaislukuparametria. Osoittimella voidaan osoittaa mihin tahansa noista neljästä funktiosta. Käyttäjää pyydetään kertomaan, mitä funktiota käytetään, jonka jälkeen osoittimeen sijoitetaan oikea arvo. Riveillä 35-36 tulostetaan kaksi kokonaislukua, kutsutaan oikeaa funktiota ja tulostetaan sitten arvot uudelleen.

Lyhennetty kutsu

Funktio-osoittimella ei tarvitse viitata uudelleen, vaikkakin niin voidaan tehdä. Niinpä, jos pFunc on osoitin funktioon, joka ottaa kokonaislukuparametrin ja palauttaa long-tyypin, ja osoittimeen sijoitetaan vastaava funktio, voidaan tuota funktiota kutsua joko koodilla

```
pFunc(x);
```

tai

```
(*pFunc) (x);
```

Nuo muodot ovat identtiset. Edellinen on lyhennetty muoto jälkimmäisestä.

Funktio-osoitintaulukko

Samalla lailla kuin esitellään int-tyyppisten osoittimien taulukko, voidaan myös esitellä taulukko, joka tallentaa osoittimia funktioihin, jotka palauttavat tietyn tyyppisen arvon ja joilla on tietty allekirjoitus. Tutki listausta 20.6.

Listaus 20.6. Funktio-osoitintaulukon käyttäminen.

```
1:  // Listing 20.6 demonstrates use of an array of pointers to functions
2:
3:  #include <iostream.h>
4:
5:  void Square (int&,int&);
6:  void Cube (int&, int&);
7:  void Swap (int&, int &);
8:  void GetVals(int&, int&);
9:  void PrintVals(int, int);
10: enum BOOL { FALSE, TRUE };
11:
12: int main()
13: {
14:   int valOne=1, valTwo=2;
15:   int choice,i;
16:   const MaxArray = 5;
17:   void (*pFuncArray[MaxArray])(int&, int&);
18:
19:   for (i=0;i<MaxArray;i++)
20:   {
21:     cout << "(1)Change Values (2)Square (3)Cube (4)Swap: ";
22:     cin >> choice;
23:     switch (choice)
24:     {
25:       case 1:pFuncArray[i] = GetVals; break;
26:       case 2:pFuncArray[i] = Square; break;
27:       case 3:pFuncArray[i] = Cube; break;
28:       case 4:pFuncArray[i] = Swap; break;
29:       default:pFuncArray[i] = 0;
30:     }
31:   }
32:
33:   for (i=0;i<MaxArray; i++)
34:   {
35:     pFuncArray[i](valOne,valTwo);
36:     PrintVals(valOne,valTwo);
37:   }
38:   return 0;
39: }
40: void PrintVals(int x, int y)
41: {
42:   cout << "x: " << x << " y: " << y << endl;
43: }
44:
45: void Square (int & rX, int & rY)
46: {
47:   rX *= rX;
48:   rY *= rY;
49: }
50:
51: void Cube (int & rX, int & rY)
52: {
```

```
53:    int tmp;
54:
55:    tmp = rX;
56:    rX *= rX;
57:    rX = rX * tmp;
58:
59:    tmp = rY;
60:    rY *= rY;
61:    rY = rY * tmp;
62:  }
63:
64:  void Swap(int & rX, int & rY)
65:  {
66:    int temp;
67:    temp = rX;
68:    rX = rY;
69:    rY = temp;
70:  }
71:
72:  void GetVals (int & rValOne, int & rValTwo)
73:  {
74:    cout << "New value for ValOne: ";
75:    cin >> rValOne;
76:    cout << "New value for ValTwo: ";
77:    cin >> rValTwo;
78:  }
```

Tulostus

```
(1)Change Values (2)Square (3)Cube (4)Swap: 1
(1)Change Values (2)Square (3)Cube (4)Swap: 2
(1)Change Values (2)Square (3)Cube (4)Swap: 3
(1)Change Values (2)Square (3)Cube (4)Swap: 4
(1)Change Values (2)Square (3)Cube (4)Swap: 2
New Value for ValOne: 2
New Value for ValTwo: 3
x: 2 y: 3
x: 4 y: 9
x: 64 y: 729
x: 729 y: 64
x: 7153 y: 4096
```

Analyysi

Riveillä 19-31 pyydetään käyttäjää valitsemaan kutsuttava funktio ja kuhunkin taulukon alkioon sijoitetaan sopivan funktion osoite. Riveillä 33-37 kutsutaan kutakin funktiota vuoron perään. Tulokset tulostetaan kunkin kutsun jälkeen.

Funktio-osoittimien vieminen muihin funktioihin

Funktio-osoittimia (kuten myös funktio-osoitintaulukoita) voidaan viedä parametreina muihin funktioihin, jotka tekevät tehtävänsä ja kutsuvat sitten oikeaa funktiota osoittimen avulla.

Saatat haluta parantaa listausta 20.6 esimerkiksi viemällä valitun funktio-osoittimen toiselle funktiolle (joka on main()-funktion ulkopuolella), joka sitten tulostaa arvot, kutsua funktiota ja tulostaa sitten arvot uudelleen. Listaus 20.7 esittää tätä muunnosta.

Listaus 20.7. Funktio-osoittimien vieminen argumentteina.

```
1:  // Listaus 20.7 Ilman funktio-osoittimia
2:
3:  #include <iostream.h>
4:
5:  void Square (int&,int&);
6:  void Cube (int&, int&);
7:  void Swap (int&, int &);
8:  void GetVals(int&, int&);
9:  void PrintVals(void (*)(int&, int&),int&, int&);
10: enum BOOL { FALSE, TRUE };
11:
12: int main()
13: {
14:   int valOne=1, valTwo=2;
15:   int choice;
16:   BOOL fQuit = FALSE;
17:
18:   void (*pFunc)(int&, int&);
19:
20:   while (fQuit == FALSE)
21:   {
22:     cout << "(0)Quit (1)Change Values (2)Square (3)Cube (4)Swap: ";
23:     cin >> choice;
24:     switch (choice)
25:     {
26:       case 1:pFunc = GetVals; break;
27:       case 2:pFunc = Square; break;
28:       case 3:pFunc = Cube; break;
29:       case 4:pFunc = Swap; break;
30:       default:fQuit = TRUE; break;
31:     }
32:     if (fQuit == TRUE)
33:       break;
34:     PrintVals ( pFunc, valOne, valTwo);
35:   }
36:
37:   return 0;
38: }
39:
40: void PrintVals( void (*pFunc)(int&, int&),int& x, int& y)
41: {
42:   cout << "x: " << x << " y: " << y << endl;
43:   pFunc(x,y);
44:   cout << "x: " << x << " y: " << y << endl;
45: }
46:
47: void Square (int & rX, int & rY)
48: {
49:   rX *= rX;
50:   rY *= rY;
51: }
52:
53: void Cube (int & rX, int & rY)
54: {
```

```
55:    int tmp;
56:
57:    tmp = rX;
58:    rX *= rX;
59:    rX = rX * tmp;
60:
61:    tmp = rY;
62:    rY *= rY;
63:    rY = rY * tmp;
64: }
65:
66: void Swap(int & rX, int & rY)
67: {
68:    int temp;
69:    temp = rX;
70:    rX = rY;
71:    rY = temp;
72: }
73:
74: void GetVals (int & rValOne, int & rValTwo)
75: {
76:    cout << "New value for ValOne: ";
77:    cin >> rValOne;
78:    cout << "New value for ValTwo: ";
79:    cin >> rValTwo;
80: }
```

Tulostus

```
(0)Quit (1)Change Values (2)Square (3)Cube (4)Swap: 1
New Value for ValOne: 2
New Value for ValTwo: 3
x: 2 y: 3
(0)Quit (1)Change Values (2)Square (3)Cube (4)Swap: 3
x: 2 y: 3
x: 8 y: 27
(0)Quit (1)Change Values (2)Square (3)Cube (4)Swap: 2
x: 8 y: 27
x: 64 y: 729
(0)Quit (1)Change Values (2)Square (3)Cube (4)Swap: 4
x: 64 y: 729
x: 729 y: 64
(0)Quit (1)Change Values (2)Square (3)Cube (4)Swap: 0
```

Analyysi

Rivillä 18 esitellään pFunc osoittimena funktioon, joka palauttaa void-arvon ja ottaa kaksi parametria (viittaukset kokonaislukuihin). Rivillä 9 esitellään PrintVals funktiona, joka ottaa kolme parametria. Ensimmäinen parametri on osoitin funktioon, joka palauttaa void-arvon ja ottaa parametreikseen kaksi viittausta kokonaislukuihin. Toinen ja kolmas parametri on kokonaislukuviittaus. Käyttäjää kehotetaan valitsemaan kutsuttava funktio ja sitten kutsutaan PrintVals-funktiota rivillä 34.

Etsi käsiisi C++ -ohjelmoija ja kysy häneltä, mitä seuraava esittely merkitsee:

```
void PrintVals(void (*) (int&, int&), int&, int&);
```

Tällaista esittelyä näkee harvoin ja luultavasti katsot käsikirjoista apua, kun sellainen tulee eteen. Mutta esittely on kuitenkin tehokas, kun juuri tuollaista rakennetta tarvitaan.

typedef funktio-osoittimien yhteydessä

Rakenne void (*) (int&, int&) on monimutkainen. Sen sijaan voit käyttää typedef-lisämäärettä yksinkertaistamaan koodia. Esittele tyyppi VPF osoittimena funktioon, joka palauttaa void-arvon ja ottaa kaksi kokonaislukuviittausta. Listaus 20.8 on edellinen listaus muokattuna siten, että käytössä on typedef-lause.

Listaus 20.8. Funktio-osoittimien tekeminen luettavammiksi typedefin avulla.

```
1:  // Listaus 20.8.  typedef
2:
3:  #include <iostream.h>
4:
5:  void Square (int&,int&);
6:  void Cube (int&, int&);
7:  void Swap (int&, int &);
8:  void GetVals(int&, int&);
9:  typedef  void (*VPF) (int&, int&) ;
10: void PrintVals(VPF,int&, int&);
11: enum BOOL { FALSE, TRUE };
12:
13: int main()
14: {
15:   int valOne=1, valTwo=2;
16:   int choice;
17:   BOOL fQuit = FALSE;
18:
19:   VPF pFunc;
20:
21:   while (fQuit == FALSE)
22:   {
23:     cout << "(0)Quit (1)Change Values (2)Square (3)Cube (4)Swap: ";
24:     cin >> choice;
25:     switch (choice)
26:     {
27:       case 1:pFunc = GetVals; break;
28:       case 2:pFunc = Square; break;
29:       case 3:pFunc = Cube; break;
30:       case 4:pFunc = Swap; break;
31:       default:fQuit = TRUE; break;
32:     }
33:     if (fQuit == TRUE)
34:       break;
35:       PrintVals ( pFunc, valOne, valTwo);
36:   }
37:   return 0;
38: }
39:
40: void PrintVals( VPF pFunc,int& x, int& y)
```

```
41: {
42: cout << "x: " << x << " y: " << y << endl;
43: pFunc(x,y);
44: cout << "x: " << x << " y: " << y << endl;
45: }
```

Tulostus

```
(0)Quit (1)Change Values (2)Square (3)Cube (4)Swap: 1
New Value for ValOne: 2
New Value for ValTwo: 3
x: 2 y: 3
(0)Quit (1)Change Values (2)Square (3)Cube (4)Swap: 3
x: 2 y: 3
x: 8 y: 27
(0)Quit (1)Change Values (2)Square (3)Cube (4)Swap: 2
x: 8 y: 27
x: 64 y: 729
(0)Quit (1)Change Values (2)Square (3)Cube (4)Swap: 4
x: 64 y: 729
x: 729 y: 64
(0)Quit (1)Change Values (2)Square (3)Cube (4)Swap: 0
```

Analyysi

Rivillä 9 käytetään typedef-lisämäärettä esittelemään VPF tyyppinä, joka on "funktio, joka palauttaa void-arvon ja ottaa kaksi parametria, jotka ovat kokonaislukuviittauksia".

Rivillä 10 esitellään PrintVals ottamaan kolme parametria: VPF ja kaksi kokonaislukuviittausta. Rivillä 19 esitellään pFunc nyt VPF-tyyppisenä.

Kun VPF-tyyppi on määritelty, ovat pFunc- ja PrintVals-esittelyt selkeämpiä.

Osoittimet jäsenfunktioihin

Tähän asti ovat kaikki funktio-osoittimet koskeneet yleisiä, muita kuin jäsenfunktioita. On myös mahdollista luoda osoittimia funktioihin, jotka ovat luokan jäseniä.

Tällainen osoitin luodaan samoin periaattein kuin tavallinenkin funktio-osoitin. Mukana on kuitenkin luokan nimi ja tuplakaksoispisteet (::). Siten, jos pFunc osoittaa luokan Shape metodiin, joka ottaa kaksi kokonaislukua ja palauttaa voidin, olisi pFunc-esittely seuraavanlainen:

```
void (Shape::*pFunc) (int, int);
```

Jäsenfunktio-osoittimia käytetään aivan samalla tavalla kuin funktio-osoittimia, paitsi että niitä voidaan kutsua vain oikean luokan oliosta käsin. Listaus 20.9 havainnollistaa jäsenfunktio-osoittimien käyttöä.

Listaus 20.9. Osoittimet jäsenfunktioihin.

```
1:  //Listaus 20.9 Osoittimet metodeihin
2:
3:  #include <iostream.h>
4:
5:  enum BOOL {FALSE, TRUE};
6:  class Mammal
7:  {
8:  public:
9:    Mammal():itsAge(1) {  }
10:   Mammal() { }
11:   virtual void Speak() const = 0;
12:   virtual void Move() const = 0;
13: protected:
14:   int itsAge;
15: };
16:
17: class Dog : public Mammal
18: {
19: public:
20:   void Speak()const { cout << "Woof!\n"; }
21:   void Move() const { cout << "Walking to heel...\n"; }
22: };
23:
24:
25: class Cat : public Mammal
26: {
27: public:
28:   void Speak()const { cout << "Meow!\n"; }
29:   void Move() const { cout << "slinking...\n"; }
30: };
31:
32:
33: class Horse : public Mammal
34: {
35: public:
36:   void Speak()const { cout << "Winnie!\n"; }
37:   void Move() const { cout << "Galloping...\n"; }
38: };
39:
40:
41: int main()
42: {
43:   void (Mammal::*pFunc)() const =0;
44:   Mammal* ptr =0;
45:   int Animal;
46:   int Method;
47:   BOOL fQuit = FALSE;
48:
49:   while (fQuit == FALSE)
50:   {
51:     cout << "(0)Quit (1)dog (2)cat (3)horse: ";
52:     cin >> Animal;
53:     switch (Animal)
54:     {
55:       case 1: ptr = new Dog; break;
56:       case 2: ptr = new Cat; break;
57:       case 3: ptr = new Horse; break;
58:       default: fQuit = TRUE; break;
59:     }
60:     if (fQuit)
```

```
61:       break;
62:
63:     cout << "(1)Speak  (2)Move: ";
64:     cin >> Method;
65:     switch (Method)
66:     {
67:
68:       case 1: pFunc = Mammal::Speak; break;
69:       default: pFunc = Mammal::Move; break;
70:     }
71:
72:     (ptr->*pFunc)();
73:
74:     delete ptr;
75:   }
76:
77:   return 0;
78: }
```

Tulostus

```
(0)Quit (1)Dog (2)Cat (3)Horse: 1
(1)Speak (2)Move: 1
Woof!
(0)Quit (1)Dog (2)Cat (3)Horse: 2
(1)Speak (2)Move: 1
Meow!
(0)Quit (1)Dog (2)Cat (3)Horse: 3
(1)Speak (2)Move: 2
Galloping
(0)Quit (1)Dog (2)Cat (3)Horse: 0
```

Analyysi

Riveillä 6-15 esitellään abstrakti tietotyyppi Mammal, jossa on kaksi puhdasta virtuaalimetodia, Speak() ja Move(). Mammal on jaettu aliluokkiin Dog, Cat ja Horse, jotka kukin korvaavat Speak()- ja Move()-metodit.

Pääohjelma pyytää käyttäjää valitsemaan luotava eläin ja sitten luodaan uusi Animal-aliluokka vapaaseen muistiin ja osoite sijoitetaan osoittimeen ptr riveillä 55-57.

Käyttäjää pyydetään sitten valitsemaan kutsuttava metodi, joka sijoitetaan pFunc-osoittimeen. Rivillä 71 kutsuu luotu olio valittua metodia käyttäen ptr-osoitinta olion käsittelyyn ja pFunc-osoitinta funktion kutsumiseen.

Lopulta kutsutaan delete-operaattoria vapauttamaan ptr-osoittimen osoittama muisti (rivi 72). Huomaa, että delete-operaattoria ei tarvitse käyttää pFunc-osoittimelle, koska se osoittaa koodiin, eikä siis muistissa olevaan kohteeseen. Itse asiassa deleten käyttäminen saisi aikaan kääntämisvirheen.

Taulukko, joka sisältää jäsenfunktio-osoittimia

Myös jäsenfunktio-osoittimia voidaan sijoittaa taulukkoon. Taulukko voidaan alustaa useiden jäsenfunktioiden osoitteilla ja niitä voidaan käsitellä siirtymällä taulukossa. Listaus 20.10 havainnollistaa tätä menettelyä.

Listaus 20.10. Taulukko, joka sisältää jäsenfunktio-osoittimia.

```
1:  //Listaus 20.10 Osoitintaulukko
2:
3:  #include <iostream.h>
4:
5:  enum BOOL {FALSE, TRUE};
6:
7:  class Dog
8:  {
9:  public:
10:   void Speak()const { cout << "Woof!\n"; }
11:   void Move() const { cout << "Walking to heel...\n"; }
12:   void Eat() const { cout << "Gobbling food...\n"; }
13:   void Growl() const { cout << "Grrrrr\n"; }
14:   void Whimper() const { cout << "Whining noises...\n"; }
15:   void RollOver() const { cout << "Rolling over...\n"; }
16:   void PlayDead() const
17:   { cout << "Is this the end of Little Caeser?\n"; }
18: };
19:
20: typedef void (Dog::*PDF)()const ;
21: int main()
22: {
23:   const int MaxFuncs = 7;
24:   PDF DogFunctions[MaxFuncs] =
25:   {  Dog::Speak,
26:   Dog::Move,
27:   Dog::Eat,
28:   Dog::Growl,
29:   Dog::Whimper,
30:   Dog::RollOver,
31:   Dog::PlayDead };
32:
33:   Dog* pDog =0;
34:   int Method;
35:   BOOL fQuit = FALSE;
36:
37:   while (!fQuit)
38:   {
39:     cout <<    "(0)Quit (1)Speak (2)Move (3)Eat (4)Growl";
40:     cout << " (5)Whimper (6)Roll Over (7)Play Dead: ";
41:     cin >> Method;
42:     if (Method == 0)
43:     {
44:       fQuit = TRUE;
45:       break;
46:     }
47:     else
48:     {
49:       pDog = new Dog;
50:      (pDog->*DogFunctions[Method-1])();
51:      delete pDog;
52:     }
53:   }
```

```
54:    return 0;
55: }
```

Tulostus

```
(0)Quit (1)Speak (2)Move (3)Eat (4)Growl (5)Whimper (6)Roll
Over (7)Play Dead: 1
Woof!
(0)Quit (1)Speak (2)Move (3)Eat (4)Growl (5)Whimper (6)Roll
Over (7)Play Dead: 4
Grrr
(0)Quit (1)Speak (2)Move (3)Eat (4)Growl (5)Whimper (6)Roll
Over (7)Play Dead: 7
Is this the end of Little Caesar?
(0)Quit (1)Speak (2)Move (3)Eat (4)Growl (5)Whimper (6)Roll
Over (7)Play Dead: 0
```

Analyysi

Riveillä 7-17 luodaan Dog-luokka, jossa on 7 jäsenfunktiota, joilla kaikilla on sama palautustyyppi ja allekirjoitus. Rivillä 20 esitellään PDF osoittimena Dogin jäsenfunktioon, joka ei ota parametreja eikä palauta arvoja ja on const-tyyppinen; eli se osoittaa Dogin jäsenfunktioiden kaltaisiin funktioihin.

Riveillä 24-31 esitellään DogFunctions-taulukko tallentamaan 7 kyseistä jäsenfunktiota ja se alustetaan noiden funktioiden osoitteilla.

Riveillä 39-41 pyydetään käyttäjää valitsemaan metodi. Ellei hän valitse nollaa, luodaan uusi Dog-olio kekoon ja kutsutaan oikeata metodia taulukosta rivillä 50. Seuraavassa on vielä yksi hieno koodirivi yrityksesi C++ -ohjelmoijille; kysy heiltä, mitä lause saa aikaan:

```
(pDog->*DogFunctions[Method-1])();
```

Lause tulee ehkä harvoin eteen, mutta kun tarvitset koodissasi taulukon, jossa on metodeita, se voi tehdä koodistasi selkeämpää.

Yhteenveto

Tässä luvussa käsiteltiin staattisten jäsenmuuttujien luomista. Kullakin luokalla (ei siis olioilla) on staattisen jäsenmuuttujan ilmentymä. On mahdollista käsitellä tuota jäsenmuuttujaa ilman oliota antamalla koko nimi, mikäli jäsenmuuttujat ovat public-tyyppiä.

Staattisia jäsenmuuttujia voidaan käyttää olioiden määrän laskentaan. Koska ne eivät ole osa oliota, niiden esittely ei vie muistia ja ne tulee määritellä ja alustaa luokan esittelyn ulkopuolella.

Staattiset jäsenfunktiot ovat osa luokkaa samalla tavalla kuin staattiset jäsenmuuttujatkin. Niitä voidaan käyttää ilman luokan oliota ja ne voivat käsitellä staattista jäsentietoa. Staattisia jäsenfunktioita ei voida käyttää ei-staattisten jäsenmuuttujien käsittelyyn, koska niillä ei ole this-osoitinta.

Koska staattisilla jäsenmuuttujilla ei ole this-osoitinta, niitä ei voida tehdä const-tyyppisiksi. Kun jäsenfunktio on const-tyyppinen, on this-osoitin const.

Luvussa käsiteltiin myös luokkia, jotka sisältävät muita luokkia. Sisällä olevalla luokalla ei ole pääsyä isäluokkansa protected-tyyppisiin jäseniin eikä se voi korvata isäluokkansa jäsenfunktioita.

Luvun yhtenä aiheena olivat myös ystäväfunktiot ja -luokat. Ystäväluokkia tulee käyttää varoen. Niiden avulla voidaan kuitenkin löytää keskitie pelkästään luokan metodien käsittelyn ja julkisiksi määrittelyn välille.

Kysymyksiä ja Vastauksia

K
Miksi käyttää staattista tietoa, kun voidaan käyttää globaalia tietoa?
V
Staattinen tieto näkyy luokassa. Siksi se on käytettävissä vain luokan olion kautta. Kutsussa käytetään luokan nimeä, jos tieto on julkista tai muutoin staattisia jäsenfunktiota. Staattinen tieto on kuitenkin luokan tyyppiä, jolloin rajoitettu pääsy ja vahva tyypitys tekevät siitä turvallisempaa kuin globaali tieto.

K
Miksi käyttää staattista jäsenfunktiota, kun käytössä ovat globaalit funktiot?
V
Staattiset jäsenfunktiot näkyvät luokassa ja niitä voidaan kutsua vain käyttäen luokan oliota tai koko nimeä (kuten ClassName::FunctionName()).

K
Miksei kaikkia luokkia tulisi tehdä kaikkien käytettävien luokkien ystäviksi?
V
Luokan tekeminen toisen luokan ystäväksi paljastaa toteutuksen yksityiskohtia ja vähentää kapselointia. Ihanteena olisi pitää kunkin luokan yksityiskohdat piilossa muilta luokilta niin pitkälle kuin mahdollista.