Osa
III

11. oppitunti

Viittaukset

Kahdessa viime luvussa opit käyttämään osoittimia kohteiden käsittelyyn vapaalla muistialueella sekä viittaamaan noihin kohteisiin epäsuorasti. Tässä luvussa käsiteltävät viittaukset ovat melkein yhtä hyviä työkaluja ja niitä on helpompi käyttää. Tämän luvun aiheita ovat:

- ❏ Mitä viittaukset ovat
- ❏ Kuinka viittaukset eroavat osoittimista
- ❏ Kuinka viittauksia luodaan ja käytetään
- ❏ Mitä viittausten rajoituksia on olemassa
- ❏ Kuinka viedään arvoja ja olioita funktioille viittauksina

Mikä on viittaus?

Viittaus on sijaisnimi. Kun luot viittauksen, alustat sen toisen kohteen nimellä. Tuosta hetkestä alkaen viittaus toimii kohteen vaihtoehtoisena nimenä ja kaikki mitä viittaukselle tehdään, tehdään myös kohteelle.

Siinä kaikki. Jotkut C++ -ohjelmoijat pitävät viittauksia osoittimina; se on väärin. Viittauksia toteutetaan usein osoittimia käyttäen ja se on vain kääntäjien kehittäjien asia; ohjelmoijan on pidettävä nuo kaksi käsitettä erillisinä.

Osoittimet ovat muuttujia, jotka tallentavat jonkun kohteen osoitteen. Viittaukset ovat sijaisnimiä toisille viittauksille.

Viittauksen luominen

Viittaus luodaan kirjoittamalla kohdeobjektin tyyppi, viittausoperaattori (&) ja viittauksen nimi. Viittauksen nimenä voi olla mikä tahansa muuttujille sopiva nimi, mutta tässä kirjassa käytetään viittausten nimien ensimmäisenä kirjaimena kirjainta r. Niinpä, jos meillä on kokonaislukumuuttuja nimeltä someInt, voit kirjoittaa viittauksen tuolle muuttujalle seuraavasti:

```
int &rSomeInt = someInt;
```

Koodi kertoo: "rSomeInt on kokonaisluvun viittaus, joka on alustettu viittaamaan muuttujaan someInt". Listaus 11.1 esittelee viittausten luomista ja käyttöä.

Huom! Huomaa, että viittaussymboli (&) on sama symboli, jota käytetään osoiteoperaattorina.

Listaus 11.1. Viittausten luominen ja käyttäminen.

```
50: //Listaus 11.1
51: // viittausten käyttö
52:
53: #include <iostream.h>
54:
55: int main()
56: {
57:   int  intOne;
58:   int &rSomeRef = intOne;
59:
60:   intOne = 5;
61:   cout << "intOne: " << intOne << endl;
62:   cout << "rSomeRef: " << rSomeRef << endl;
63:
64:   rSomeRef = 7;
65:   cout << "intOne: " << intOne << endl;
66:   cout << "rSomeRef: " << rSomeRef << endl;
67:   return 0;
68: }
```

Tulostus

```
intOne: 5
rSomeRef: 5
intOne: 7
rSomeRef: 7
```

Analyysi

Rivillä 8 esitellään kokonaislukumuuttuja intOne. Rivillä 9 esitellään viittaus int-tyyppiin, rSomeRef, joka alustetaan viittaamaan muuttujaan intOne. Jos esittelet viittauksen, mutta et alusta sitä, saat kääntäjän virheilmoituksen. Viittaukset tulee alustaa.

Rivillä 11 sijoitetaan muuttujaan intOne arvo 5. Riveillä 12-13 tulostetaan sitten intOne- ja rSomeRef-arvot ja ne ovat tietenkin samat.

Rivillä 15 sijoitetaan taas arvo 7 viittaukseen rSomeRef. Vastaava tulostus toteutetaan sitten riveillä 16-17.

Osoiteoperaattorin käyttö viittausten yhteydessä

Jos kysyt viittauksen osoitetta, se palauttaa kohteensa osoitteen. Se juuri onkin luontaista viittaukselle, sehän on kohteensa sijaisnimi. Listaus 11.2 havainnollistaa tätä.

Listaus 11.2. Viittauksen osoite.

```
1:  //Listaus 11.2
2:  // viittausten käyttö
3:
4:  #include <iostream.h>
5:
6:  int main()
7:  {
8:    int  intOne;
9:    int &rSomeRef = intOne;
10:
11:   intOne = 5;
12:   cout << "intOne: " << intOne << endl;
13:   cout << "rSomeRef: " << rSomeRef << endl;
14:
15:   cout << "&intOne: "  << &intOne << endl;
16:   cout << "&rSomeRef: " << &rSomeRef << endl;
17:
18:   return 0;
19: }
```

Tulostus

```
intOne: 5
rSomeRef: 5
&intOne: 0x3500
&rSomeRef: 0x3500
```

Analyysi

Viittaus rSomeRef alustetaan muuttujalla intOne. Nyt tulostetaan näiden muuttujien osoitteet ja ne ovat samat. Itse viittauksen osoitteeseen ei päästä käsiksi ja se onkin merkityksetöntä. Viittaukset alustetaan luomisen yhteydessä ja ne toimivat aina kohteidensa synonyymeinä, myös osoitteita käsiteltäessä.

Olettakaamme, että meillä on luokka nimeltä President ja luomme luokan olion seuraavasti:

```
President William_Jefferson_Clinton;
```

Sen jälkeen esittelemme viittauksen President-luokkaan ja alustamme sen tuolla oliolla:

```
President &Bill_Clinton = William_Jefferson_Clinton;
```

President-luokan edustajia on vain yksi; molemmat tunnisteet viittaavat saman luokan olioon. Jokainen Bill_Clinton-toiminto kohdistetaan myös oliolle William_Jefferson_Clinton.

Varmista aina, että erotat toisistaan osoiteoperaattorina käytetyn &-symbolin ja viittauksissa käytetyn &-symbolin. Vertaa rivejä 9 ja 15-16.

Yleensä viittauksia käytettäessä ei tarvita osoiteoperaattoria. Viittausta käytetään aivan kuin kohdemuuttujaa (katso rivi 13).

Jopa kokeneet C++ -ohjelmoijat voivat olla epävarmoja siitä, mitä tapahtuu, kun viittaukseen yritetään sijoittaa uusi kohde. Uuden arvon sijoittaminen osoittautuu uuden arvon sijoittamiseksi kohteeseen. Listaus 11.3 havainnollistaa tätä tosiseikkaa.

Listaus 11.3. Viittaukseen sijoittaminen.

```
1:  //Listaus 11.3
2:  //uudelleen viittaus
3:
4:  #include <iostream.h>
5:
6:  int main()
7:  {
8:    int  intOne;
9:    int &rSomeRef = intOne;
10:
11:   intOne = 5;
12:   cout << "intOne:\t" << intOne << endl;
13:   cout << "rSomeRef:\t" << rSomeRef << endl;
14:   cout << "&intOne:\t"  << &intOne << endl;
15:   cout << "&rSomeRef:\t" << &rSomeRef << endl;
16:   int intTwo = 8;
17:   rSomeRef = intTwo;  // not what you think!
18:   cout << "\nintOne:\t" << intOne << endl;
19:   cout << "intTwo:\t" << intTwo << endl;
20:   cout << "rSomeRef:\t" << rSomeRef << endl;
```

```
21:    cout << "&intOne:\t"  << &intOne << endl;
22:    cout << "&intTwo:\t"  << &intTwo << endl;
23:    cout << "&rSomeRef:\t" << &rSomeRef << endl;
24:    return 0;
25: }
```

Tulostus

```
intOne:     5
rSomeRef        5
&intOne:        0x213e
&rSomeRef   0x213e

intOne:     8
intTwo      8
rSomeRef        8
&intOne:        0x213e
&intTwo     0x2130
&rSomeRef   0x213e
```

Analyysi

Jälleen kerran esitellään kokonaislukumuuttuja ja int-viittaus riveillä 8 ja 9. Muuttujaan sijoitetaan arvo 5 rivillä 11 ja arvo sekä osoite tulostetaan riveillä 12-15.

Rivillä 17 luodaan uusi muuttuja intTwo ja alustetaan se arvolla 8. Rivillä 18 yrittää ohjelmoija sijoittaa viittaukseen rSomeRef uuden arvon eli asettamaan sen muuttujan intTwo sijaisnimeksi. Näin ei kuitenkaan tapahdu. Viittaus toimii edelleenkin muuttujan intOne sijaisnimenä eli sijoitus on yhdenmukainen seuraavan koodin kanssa:

```
intOne = intTwo;
```

Edellä kerrottu vahvistetaan rivien 19-21 tulostuksissa. Rivien 22-24 osoitteiden tulostuksessa nähdään vielä, että viittaus rSomeRef viittaa edelleenkin muuttujaan intOne eikä muuttujaan intTwo.

Tee/Älä tee

Käytä viittauksia luomaan kohteiden sijaisia.
Alusta kaikki viittaukset.
Älä yritä sijoittaa viittaukseen uutta viittausarvoa.
Älä sotke osoiteoperaattoria viittausoperaattoriin.

Mihin voidaan viitata?

Mihin tahansa kohteeseen, myös käyttäjän määrittelemään, voidaan luoda viittaus. Huomaa, että viittaus luodaan oliolle, ei siis luokalle. Et voi siis kirjoittaa:

```
int &rIntRef = int; // väärin
```

Viittaus on alustettava tietyllä kokonaislukumuuttujalla, eli:

```
int howBig = 200;
int &rInteRef = howBig;
```

Samalla lailla luodaan viittaus Cat-olioon:

```
CAT &rCatRef = CAT;   // väärin
```

Viittaus on alustettava tietyllä CAT-oliolla:

```
CAT Frisky;
CAT &rCatRef = Frisky;
```

Olioviittauksia käytetään aivan kuin itse olioitakin. Jäsenmuuttujia ja metodeita käsitellään käyttäen normaalia jäsenoperaattoria (.) ja viittaus toimii olion sijaisnimenä.

Null-osoittimet ja -viittaukset

Kun osoittimia ei alusteta tai kun ne tuhotaan, ne tulisi asettaa null-osoittimiksi. Näin ei kuitenkaan tehdä viittausten yhteydessä. Viittaus ei voi olla null, eikä ohjelma, jossa käytetään viittauksia null-kohteisiin, toimikaan oikein. Tällainen ohjelma voi toimiakin jonkun aikaa, mutta se saattaa myös tuhota kaikki levytiedostosi.

Useimmat kääntäjät tukevat null-kohteita ja kaatuvat vain yritettäessä käyttää tuota kohdetta jollakin tavalla. Null-kohteita ei kannata kuitenkaan käyttää, sillä ohjelmassa voi esiintyä hämäriä virheitä silloin, kun ohjelma siirretään toiseen ympäristöön tai kääntäjä vaihdetaan toiseen.

Funktion argumenttien vieminen viittauksina

Luvussa 5, "Funktiot", kerrottiin, että funktioilla on kaksi rajoitetta: argumentit viedään arvoina ja return-lause voi palauttaa vain yhden arvon.

Kun arvot viedään funktioille viittauksina, päästään noista molemmista rajoituksista. Viittauksena vieminen toteutetaan C++ -kielessä kahdella tavalla: osoittimia käyttäen ja viittauksia käyttäen. Syntaksit poikkeavat toisistaan, mutta lopputulos on sama: kopion sijaan funktiolle viedään todellinen, alkuperäinen kohde.

Kohteen vieminen viittauksena mahdollistaa sen, että funktio voi muokata viitattua kohdetta.

Listaus 11.4 esittelee vaihtofunktion ja siinä parametrit viedään funktiolle arvoina.

Listaus 11.4. Funktion parametrit viedään arvoina.

```
26: //Listaus 11.4 Arvoparametrit
27:
28: #include <iostream.h>
29:
30: void swap(int x, int y);
31:
32: int main()
33: {
34:   int x = 5, y = 10;
35:
36:   cout << "Main. Before swap, x: " << x << " y: " << y << "\n";
37:   swap(x,y);
38:   cout << "Main. After swap, x: " << x << " y: " << y << "\n";
39:   return 0;
40: }
41:
42: void swap (int x, int y)
43: {
44:   int temp;
45:
46:   cout << "Swap. Before swap, x: " << x << " y: " << y << "\n";
47:
48:   temp = x;
49:   x = y;
50:   y = temp;
51:
52:   cout << "Swap. After swap, x: " << x << " y: " << y << "\n";
53:
54: }
```

Tulostus

```
Main. Before swap, x: 5 y: 10
Swap. Before swap, x: 5 y: 10
Swap. After swap, x: 10 y: 5
Main. After swap, x: 5 y: 10
```

Analyysi

Ohjelmassa alustetaan kaksi muuttujaa main()-funktiossa ja viedään sitten muuttujat swap()-funktiolle, joka vaihtaa niiden arvot. Mutta, kun tuloksia tutkitaan main()-funktiossa, muuttujien arvot ovat edelleenkin samat!

Ongelma on siinä, että muuttujat x ja y viedään arvoina swap()-funktiolle. Tällöin funktio tekee niiden kopiot ja käsittelee kopioita eikä alkuperäisiä arvoja. Parametrit on siis vietävä viittauksina.

Ongelma voidaan ratkaista kahdella tavalla: swap()-funktion parametrit muutetaan alkuperäisten arvojen osoittimiksi tai viedään parametrit alkuperäisten arvojen viittauksina.

swap()-funktion käyttö osoittimien kanssa

Kun viet funktiolle parametrina osoittimen, viet sille kohteen osoitteen, jolloin funktio voi muokata tuossa osoitteessa olevaa arvoa. Jotta swap()-funktio voisi muuttaa sille vietyjä arvoja, on arvot vietävä osoittimien muodossa ja funktio itse on esiteltävä ottamaan parametreikseen kaksi int-osoitinta. Kun sitten viitataan osoittimilla uudelleen, vaihtavat x ja y paikkaansa. Listaus 11.5 esittelee tätä menettelyä.

Listaus 11.5. Osoittimien vienti.

```
1:  //Listaus 11.4 Osoittimien vienti
2:
3:  #include <iostream.h>
4:
5:  void swap(int *x, int *y);
6:
7:  int main()
8:  {
9:    int x = 5, y = 10;
10:
11:   cout << "Main. Before swap, x: " << x << " y: " << y << "\n";
12:   swap(&x,&y);
13:   cout << "Main. After swap, x: " << x << " y: " << y << "\n";
14:   return 0;
15: }
16:
17: void swap (int *px, int *py)
18: {
19:   int temp;
20:
21:   cout << "Swap. Before swap, *px: " << *px << " *py: " << *py << "\n";
22:
23:   temp = *px;
24:   *px = *py;
25:   *py = temp;
26:
27:   cout << "Swap. After swap, *px: " << *px << " *py: " << *py << "\n";
28:
29: }
```

Tulostus

```
Main. Before swap, x: 5 y: 10
Swap. Before swap, *px: 5 *py: 10
Swap. After swap, *px: 10 *py: 5
Main. After swap, x: 10 y: 5
```

Analyysi

Se onnistui! Rivillä 5 on esitelty swap() parametreinaan kaksi int-osoitinta kahden int-muuttujan sijaan. Kun funktiota kutsutaan rivillä 12, sille viedään argumentteina muuttujien x ja y osoitteet.

Rivillä 19 esitellään swap()-funktiossa paikallinen muuttuja temp; sen ei tarvitse olla osoitin. Siihen tallennetaan *px-arvo (eli kutsuvan funktion x-arvo) funktion ajon ajaksi. Kun funktio päättyy, on temp tarpeeton.

Rivillä 23 sijoitetaan temp-muuttujaan px-arvo. Rivillä 24 sijoitetaan px-arvo osoitinmuuttujaan py. Rivillä 25 temp-muuttujan arvo (eli px:n alkuperäinen arvo) sijoitetaan py-muuttujaan.

Tuloksena vaihtavat kutsuvan funktion muuttujien arvot paikkaansa.

swap()-funktion toteutus viittauksin

Edellinen ohjelma toimii, mutta swap()-funktion syntaksi on hieman monimutkainen kahdella tavalla. Ensiksikin osoittimien toistuva uudelleenviittaus lisää virhealttiutta ja tekee koodista vaikealukuista. Toiseksi muuttujien osoitteiden vieminen swap()-funktiolle paljastaa swap()-funktion sisäisen toiminnan.

C++ -kielen yhtenä tavoitteena on kätkeä funktion sisäinen toiminta siten, ettei käyttäjän tarvitse huolehtia siitä, kuinka jokin funktio toimii. Osoittimien vieminen asettaa painopisteen kutsuvaan funktioon eikä kutsuttavaan funktioon. Listaus 11.6 sisältää muutetun swap()-funktion, jossa käytetään viittauksia.

Listaus 11.6. swap() viittauksia käyttäen.

```
1:  //Listaus 11.4 Viittausten vienti
2:
3:  #include <iostream.h>
4:
5:  void swap(int &x, int &y);
6:
7:  int main()
8:  {
9:    int x = 5, y = 10;
10:
11:   cout << "Main. Before swap, x: " << x << " y: " << y << "\n";
12:   swap(x,y);
13:   cout << "Main. After swap, x: " << x << " y: " << y << "\n";
14:   return 0;
15: }
16:
17:   void swap (int &rx, int &ry)
18: {
19:   int temp;
20:
21:   cout << "Swap. Before swap, rx: " << rx << " ry: " << ry << "\n";
22:
23:   temp = xr;
24:   rx = ry;
25:   ry = temp;
26:
27:   cout << "Swap. After swap, rx: " << rx << " ry: " << ry << "\n";
28:
29: }
```

Tulostus

```
Main. Before swap, x: 5 y: 10
Swap. Before swap, x: 5 y: 10
Swap. After swap, x: 10 y: 5
Main. After swap, x: 10 y: 5
```

Analyysi

Rivillä 9 esitellään kaksi muuttujaa kuten aiemmassa esimerkissämmekin ja niiden arvot tulostetaan rivillä 11. Rivillä 12 kutsutaan swap()-funktiota: parametreina ovat x ja y, eivätkä niiden osoitteet. Kutsuva funktio yksinkertaisesti vie muuttujat parametreina.

Kun swap()-funktiota kutsutaan, ohjelman suoritus siirtyy riville 18, jossa muuttujat identifioidaan viittauksina. Niiden arvot tulostetaan rivillä 21, mutta mitään erikoisoperaattoreita ei tarvita. Kyseessä ovat alkuperäisten arvojen sijaisnimet, joita voidaan käyttää sellaisinaan.

Riveillä 23-25 vaihdetaan arvojen paikkaa ja tulostus tapahtuu rivillä 27. Ohjelman suoritus palaa sitten kutsuvaan funktioon ja rivillä 13 tulostetaan arvot uudelleen. Koska swap()-funktion parametrit ovat viittauksia, viedään arvot main()-funktiosta viittauksina ja siten niiden arvot vaihtavat paikkaansa todellisuudessa.

Viittaukset tarjoavat kätevän ja helpon keinon käsitellä tavallisia muuttujia, ja antavat samalla mahdollisuuden viittauksena viemiseen kuten osoittimetkin.

Funktion otsikon ja prototyypin ymmärtäminen

Listauksessa 11.7 oleva swap() käyttää osoittimia ja listauksessa 11.8 oleva swap() taas viittauksia. Viittausten käyttäminen on helpompaa ja koodi on selkeämpää, mutta mistä kutsuva funktio tietää, viedäänkö arvot viittauksina vai arvoina? swap()-funktion käyttäjänä on ohjelmoijan taattava, että funktio todellakin vaihtaa parametriensa arvoja.

Avainasemassa on funktion prototyyppi eli sen esittely. Prototyypin (esittelyt ovat yleensä otsikkotiedostossa) parametreja tutkimalla ohjelmoija tietää, että arvot viedään funktiolle viittauksina, jolloin ne vaihtavat paikkaansa oikein.

Jos swap() olisi luokan jäsenfunktio, antaisi luokan esittely saman informaation.

C++ -kielessä luokkien ja funktioiden asiakkaat nojautuvat otsikkotiedostoon; se toimii luokan tai funktion käyttöliittymänä. Todellinen

toteutus on kätketty asiakkaalta. Näin ohjelmoija voi keskittyä itse ongelman ratkaisemiseen ja olla välittämättä siitä, kuinka kohdefunktio tai -luokka toimii.

Useiden arvojen palauttaminen

Aiemmin kerrottiin, että funktiot voivat palauttaa vain yhden arvon. Entä, jos sinun pitää palauttaa kaksi arvoa funktiosta? Eräs keino ratkaista tämä ongelma on viedä kaksi kohdetta funktioon, viittauksina. Funktio voi sitten täyttää kohteet oikeilla arvoilla. Koska viittauksena vieminen mahdollistaa funktion muuttaa alkuperäisiä kohteita, voi funktio itse asiassa palauttaa kaksi tietoa. Tämä lähestymistapa ohittaa funktion palauttaman arvon, jota voidaan sitten käyttää virheiden raportointiin.

Edellä kerrottu voidaan toteuttaa taas kerran joko viittauksin tai osoittimin. Listaus 11.7 esittelee funktion, joka palauttaa kolme arvoa, kaksi osoitinparametrein ja yhden palautusarvona.

Listaus 11.7. Arvojen palauttaminen osoittimien avulla.

```
1:  //Listaus 11.7
2:  // Useiden arvojen palauttaminen
3:
4:  #include <iostream.h>
5:
6:  typedef unsigned long ULONG;
7:
8:  long Factor(ULONG, ULONG*, ULONG*);
9:
10: int main()
11: {
12:   ULONG number, squared, cubed;
13:   long error;
14:
15:   cout << "Enter a number (0 - 20): ";
16:   cin >> number;
17:
18:   error = Factor(number, &squared, &cubed);
19:
20:   if (!error)
21:   {
22:     cout << "number: " << number << "\n";
23:     cout << "square: " << squared << "\n";
24:     cout << "cubed: "  << cubed   << "\n";
25:   }
26:   else
27:     cout << "Error encountered!!\n";
28:   return 0;
29: }
30:
31: long Factor(ULONG n, ULONG *pSquared, ULONG *pCubed)
32: {
33:   long Value;
34:   if (n > 20)
35:     Value = 1L;
36:   else
37:   {
```

```
38:      *pSquared = n*n;
39:      *pCubed = n*n*n;
40:      Value = 0L;
41:    }
42:    return Value;
43: }
```

Tulostus

```
Enter a number (0-20): 3
number: 3
square: 9
cubed: 27
```

Analyysi

Rivillä 12 määritellään muuttujat number, squared ja cubed USHORT-tyyppisinä. Käyttäjän antama arvo sijoitetaan muuttujaan number. Tuo arvo sekä muuttujien squared ja cubed osoitteet viedään funktiolle Factor().

Factor() tutkii ensimmäisen parametrin, joka viedään arvona. Jos se on suurempi kuin 20 (funktion käsittelemä maksimiarvo), sijoitetaan Value-muuttujaan virhearvo. Huomaa, että Factor()-funktion palautusarvona on joko tuo virhearvo tai arvo 0, joka kertoo onnistumisesta. Huomaa myös, että funktio palauttaa tuon arvon rivillä 42.

Todelliset tarvittavat arvot, number-arvon neliö ja kuutio, palautetaan muulla tavalla, ei return-lauseella. Ne palautetaan muuttamalla funktiolle vietyjä osoittimia.

Riveillä 38-39 sijoitetaan osoittimiin niiden palautusarvot. Rivillä 40 sijoitetaan Value-muuttujaan onnistumista kuvaava arvo ja se palautetaan rivillä 42.

Eräs parannus ohjelmaan saattaisi olla seuraava esittely:

```
enum ERROR_VALUE { SUCCESS, FAILURE};
```

Sitten ohjelma voisi arvojen 0 tai 1 sijaan palauttaa joko SUCCESS tai FAILURE.

Arvojen palauttaminen viittauksina

Vaikka listaus 11.7 toimiikin, siitä tulee luettavampaa ja ylläpidettävämpää, kun osoittimien sijaan käytetään viittauksia. Listaus 11.8 sisältää saman ohjelman muutettuna käyttämään viittauksia; siinä on myös edellä annettu, selkeämpi virhe- tai onnistumistiedon palauttaminen.

Listaus 11.8. Edellinen esimerkki viittauksin.

```
1:  //Listaus 11.8
2:  // Returning multiple values from a function
3:  // using references
4:
5:  #include <iostream.h>
6:
7:  typedef unsigned long ULONG;
8:  enum ERR_CODE { SUCCESS, ERROR };
9:
10: ERR_CODE Factor(ULONG, ULONG&, ULONG&);
11:
12: int main()
13: {
14:   ULONG number, squared, cubed;
15:   ERR_CODE result;
16:
17:   cout << "Enter a number (0 - 20): ";
18:   cin >> number;
19:
20:   result = Factor(number, squared, cubed);
21:
22:   if (result == SUCCESS)
23:   {
24:     cout << "number: " << number << "\n";
25:     cout << "square: " << squared << "\n";
26:     cout << "cubed: "  << cubed   << "\n";
27:   }
28:   else
29:     cout << "Error encountered!!\n";
30:   return 0;
31: }
32:
33: ERR_CODE Factor(ULONG n, ULONG &rSquared, ULONG &rCubed)
34: {
35:   if (n > 20)
36:     return ERROR;   // simple error code
37:   else
38:   {
39:     rSquared = n*n;
40:     rCubed = n*n*n;
41:     return SUCCESS;
42:    }
43: }
```

Tulostus

```
Enter a number (0-20): 3
number: 3
square: 9
cubed: 27
```

Analyysi

Listaus 11.8 on samanlainen kuin listaus 11.7 lukuunottamatta kahta poikkeusta. ERR_CODE-toteutus parantaa virheen raportointia (rivit 36, 41) sekä virheen käsittelyä (rivi 22).

Suurempi muutos on kuitenkin siinä, että Factor() on nyt esitelty ottamaan parametreikseen viittaukset squared- ja cubed-muuttujiin osoittimien sijaan. Nyt näiden parametrien muokkaaminen on yksinkertaisempaa ja helpompaa ymmärtää.

Yhteenveto

Tämän luvun myötä opit viittaukset ja niiden erot osoittimiin. Näit, että viittaukset on alustettava viittaamaan olemassa olevaan kohteeseen eikä niitä voida laittaa viittaamaan uudelleen mihinkään muuhun. Jokainen viittaukselle tehty toiminto tehdään itse asiassa viittauksen kohteelle. Tämän todentaa se, että viittauksen osoite palauttaa kohteen osoitteen.

Kysymyksiä ja Vastauksia

K
Miksi viittauksia kannattaisi käyttää, kun osoittimet pystyvät tekemään kaiken sen, minkä viittauksetkin?
V
Viittauksia on helpompi käyttää ja ymmärtää. Epäsuoruus on kätkettynä eikä toistuvaan muuttujan uudelleen viittaamiseen ole tarvetta.

K
Miksi osoittimia tarvitaan, kun viittaukset ovat helpompia?
V
Viittaukset eivät voi olla null-viittauksia eikä niihin voida sijoittaa uutta kohdetta. Osoittimet tarjoavat enemmän joustavuutta ollen samalla hieman vaikeampia käyttää.