Osa
II

8. oppitunti

Kehittynyt ohjelman kulku

Toimintojen toteuttaminen ohjelmissa vaatii usein haarautumisia ja silmukoita. Tässä luvussa tutustummekin seuraaviin asioihin:

- ❏ Mitä silmukat ovat ja kuinka niitä käytetään
- ❏ Kuinka erilaisia silmukoita kehitetään
- ❏ Vaihtoehto sisäkkäisille if … else -rakenteille

Silmukointi

Monet ohjelmointiongelmat ratkaistaan käsittelemällä samaa tietoa toistuvasti.

Uusi käsite Iterointi tarkoittaa sitä, että sama toiminto toteutetaan uudelleen ja uudelleen, kunnes ollaan tavoitetuloksessa. Iteroinnin periaate toteutuu silmukassa.

Silmukoinnin alku - goto-rakenne

Ohjelmoinnin alkupäivinä silmukat sisälsivät otsikoidun ohjelmarivin, joitakin ohjelmalauseita ja hypyn.

C++ -kielessä otsikko on nimi, jota seuraa kaksoispiste (:). Otsikko sijoitetaan ohjelmalauseen vasemmalle puolelle ja hyppy kyseiselle riville toteutetaan goto-komennolla, jota seuraa otsikon nimi. Listaus 8.1 havainnollistaa tätä menettelyä.

Listaus 8.1. Silmukointi goto-lauseella.

```
45: // Listaus 8.1
46: // goto-haarautuminen
47:
48: #include <iostream.h>
49:
50: int main()
51: {
52:   int counter = 0;      // alusta counter
53:   loop:  counter ++;           // loopin alku
54:   cout << "counter: " << counter << "\n";
55:   if (counter < 5)            // testaa arvo
56:   goto loop;                 // hypää alkuun
57:
58:   cout << "Complete. counter: " << counter << ".\n";
59:   return 0;
60: }
```

Tulostus

```
Counter: 1
Counter: 2
Counter: 3
Counter: 4
Counter: 5
Complete. Counter: 5
```

Analyysi

Rivillä 8 alustetaan counter-muuttuja arvolla 0. Otsikko, loop, sijaitsee rivillä 9, silmukan alussa. Silmukassa kasvatetaan counter-muuttujaa ja sen arvo tulostetaan. Muuttujan arvo testataan rivillä 11. Jos se on pienempi kuin 5, on if-ehto tosi ja goto-komento suoritetaan. Tällöin hypätään takaisin riville 9. Silmukkaa suoritetaan, kunnes counterin arvo on 5, jolloin silmukasta poistutaan ja tulostetaan viimeinen viesti.

Miksi goto-komentoa ei juuri käytetä?

Goto-komentoa ei yleensä käytetä C++ -kielessä ja syykin on ilmeinen. Goto-komento sallii hypyn mihin tahansa koodissa, eteenpäin tai taaksepäin. Holtiton goto-komennon käyttö on aiheuttanut hajanaista, rikkonaista ja vaikealukuista koodia, joka tunnetaan spagettikoodina.

Välttääkseen goto-komennon käyttöä voi ohjelmoija käyttää kehittyneempiä ja hallittavampia silmukkarakenteita for, while ja do…while. Noilla rakenteilla voidaan kehittää ohjelmia, jotka ovat ymmärrettävämpiä eikä goto-lausetta tarvita.

While-silmukat

While-silmukalla voidaan toistaa ohjelmalauseita, niin kauan kuin alkuehto on tosi. Listauksessa 8.1 kasvatettiin counteria, kunnes sen arvo oli 5. Listauksessa 8.2 on sama ohjelma kirjoitettu uudelleen käyttämään while-silmukkaa.

Listaus 8.2. while-silmukka.

```
1:  // Listaus 8.2
2:  // while-silmukointi
3:
4:  #include <iostream.h>
5:
6:  int main()
7:  {
8:    int counter = 0;              // ehdon alustus
9:
10:   while(counter < 5)     // testi
11:   {
12:     counter++;              // loopin runko
13:     cout << "counter: " << counter << "\n";
14:   }
15:
16:   cout << "Complete. counter: " << counter << ".\n";
17:   return 0;
18: }
```

Tulostus

```
Counter: 1
Counter: 2
Counter: 3
Counter: 4
Counter: 5
Complete. Counter: 5
```

Analyysi

Tämä yksinkertainen ohjelma esittelee while-silmukan perusperiaatteet. Ehto testataan ja jos se on tosi, suoritetaan while-silmukka. Tässä tapauksessa ehto (onko counterin arvo pienempi kuin 5) testataan rivillä 10. Jos ehto on tosi, suoritetaan silmukan runko. Rivillä 12 kasvatetaan counterin arvoa ja rivillä 13 tulostetaan sen arvo. Kun rivin 10 ehto on epätosi (counter ei ole pienempi kuin 5), ohitetaan koko while-silmukan runko (rivit 11-14). Ohjelman suoritus siirtyy riville 15.

Monimutkaisempia while-rakenteita

While-silmukassa testattava alkuehto voi olla millainen C++ -lauseke tahansa. Se voi sisältää myös loogisia osia (ja, tai, ei). Listaus 8.3 esittelee monimutkaisemman while-silmukan.

Listaus 8.3. Monipuolinen while-silmukka.

```
1:  // Listaus 8.3
2:  // Laajempi while-käyttö
3:
4:  #include <iostream.h>
5:
6:  int main()
7:  {
8:    unsigned short small;
9:    unsigned long  large;
10:   const unsigned short MAXSMALL=65535;
11:
12:   cout << "Enter a small number: ";
13:   cin >> small;
14:   cout << "Enter a large number: ";
15:   cin >> large;
16:
17:   cout << "small: " << small << "...";
18:
19: // kullakin kierroksella testataan 3 ehtoa
20:   while (small < large && large > 0 && small < MAXSMALL)
21:
22:   {
23:     if (small % 5000 == 0)  // write a dot every 5k lines
24:       cout << ".";
25:
26:     small++;
27:
28:     large-=2;
29:   }
30:
31:   cout << "\nSmall: " << small << " Large: " << large << endl;
32:   return 0;
33: }
```

Tulostus

```
Enter a small number: 2
Enter a large number: 100000
Small: 2…….
Small: 33335 Large: 33334
```

Analyysi

Kyseessä on peliohjelma. Siinä annetaan kaksi lukua, pieni ja suuri. Pientä lukua kasvatetaan aina yhdellä ja suurta taas vähennetään kahdella. Tavoitteena on arvata, milloin luvut kohtaavat.

Riveillä 12-15 syötetään luvut. Rivi 20 alustaa while-silmukan, joka toistuu, kunnes seuraavat ehdot toteutuvat:

- ❑ Pieni luku ei ole suurempi kuin suuri luku
- ❑ Suuri luku ei ole negatiivinen
- ❑ Pieni luku ei ylitä pienen kokonaisluvun (MAXSMALL) kokoa

Rivillä 22 lasketaan small-muuttujan arvo modulo 5000. Lause ei muuta small-muuttujan arvoa vaan palauttaa arvon 0 silloin, kun small on täsmälleen luvun 5000 monikerta. Aina, kun näin on, tulostetaan piste (.) näytölle kertomaan ohjelman edistymisestä. Rivillä 25 kasvatetaan small-arvoa ja rivillä 27 vähennetään large-arvoa kahdella.

Kun mikä tahansa while-ehdoista on epätosi, silmukka ohitetaan ja suoritus siirtyy riville 27.

Silmukan keskeyttäminen: continue ja break

Ohjelmissa joudutaan joskus palaamaan while-silmukan alkuun ennen kaikkien silmukan runkolauseiden suorittamista. Tämä voidaan toteuttaa continue-lauseella.

Joskus taas joudutaan poistumaan silmukasta ennen kaikkien alkuehtojen toteutumista. Silmukasta poistuminen voidaan toteuttaa break-komennolla.

Listaus 8.4 esittelee näiden lauseiden käyttöä. Nyt peliohjelma on monipuolisempi. Käyttäjää pyydetään antamaan pieni (small) ja suuri (large) luku, ohitusluku (skip) sekä tavoiteluku (target). Pientä lukua kasvatetaan yhdellä ja suurta vähennetään kahdella. Vähentäminen ohitetaan aina, kun pieni luku on skip-luvun monikerta. Peli päättyy, jos pienestä luvusta tulee suurempi kuin suuri luku. Jos suuri luku saavuttaa tavoitearvon, tulostetaan viesti ja peli päättyy.

Käyttäjän tavoitteena on asettaa suurelle luvulle tavoitearvo, joka lopettaa pelin.

Listaus 8.4. break ja continue.

```
1:  // Listaus 8.4
2:  // break ja continue
3:
4:  #include <iostream.h>
5:
6:  int main()
7:  {
8:    unsigned short small;
9:    unsigned long  large;
10:   unsigned long  skip;
11:   unsigned long target;
12:   const unsigned short MAXSMALL=65535;
13:
```

```
14:    cout << "Enter a small number: ";
15:    cin >> small;
16:    cout << "Enter a large number: ";
17:    cin >> large;
18:    cout << "Enter a skip number: ";
19:    cin >> skip;
20:    cout << "Enter a target number: ";
21:    cin >> target;
22:
23:    cout << "\n";
24:
25:  // 3 ehtoa
26:    while (small < large && large > 0 && small < MAXSMALL)   // Maxsmall = 65535
27:
28:    {
29:
30:      small++;
31:
32:      if (small % skip == 0)  // hypätään kasvatus
33:      {
34:        cout << "skipping on " << small << endl;
35:        continue;
36:      }
37:
38:      if (large == target)    // onko oikein
39:      {
40:        cout << "Target reached!";
41:        break;
42:      }
43:
44:      large-=2;
45:    }                  // whilen loppu
46:
47:  cout << "\nSmall: " << small << " Large: " << large << endl;
48:  return 0;
49:  }
```

Tulostus

```
Enter a small number: 2
Enter a large number: 2
Enter a skip number: 2
Enter a target number: 2
Skipping on 4
Skipping on 8
Small: 10 Large: 8
```

Analyysi

Esimerkkiajossamme käyttäjä hävisi pelin. Pienestä luvusta tuli suurempi ennen kuin tavoitearvo 6 saavutettiin.

Rivillä 26 testataan while-ehto. Jos small-luku on pienempi kuin large, large on suurempi kuin 0 eikä small ole ylittänyt pienen kokonaisluvun maksimiarvoa, suoritetaan while-silmukan runko.

Rivillä 31 lasketaan small-arvo modulo skip-arvo. Jos small on skip-arvon monikerta, suoritetaan continue-lause ja suoritus palaa silmukan alkuun riville 26. Tällöin ohitetaan target-arvon testi ja large-arvon vähentäminen.

Rivillä 37 testataan target-arvo suhteessa large-arvoon. Jos luvut ovat samoja, käyttäjä on voittanut. Tällöin tulostetaan viesti ja poistutaan silmukasta break-lauseella. Hyppy tapahtuu riville 45.

Sekä continue- että break-lausetta tulee käyttää varoen. Ne ovat seuraavaksi vaarallisimpia goto-lauseen jälkeen ja vieläpä samasta syystä. Ohjelmia, jotka muuttavat yhtä äkkiä suuntaansa, on vaikea ymmärtää.

While(1) -silmukat (ikuiset silmukat)

While-silmukassa testattava ehto voi olla mikä tahansa soveltuva C++ -ilmaus. Silmukkaa suoritetaan niin kauan kuin ehto on tosi. On mahdollista luoda silmukka, joka ei pääty koskaan käyttämällä ehtona lukua 1. Koska 1 on aina tosi, on silmukka ikuinen ellei break-lausetta ole mukana. Listaus 8.5 esittelee laskemista kymmeneen tällaisella rakenteella.

Listaus 8.5. while(1) -silmukka.

```
1:  // Listaus 8.5
2:  // while-ehto tosi
3:
4:  #include <iostream.h>
5:
6:  int main()
7:  {
8:    int counter = 0;
9:
10:   while (1)
11:   {
12:     counter ++;
13:     if (counter > 10)
14:       break;
15:   }
16:   cout << "counter: " << counter << "\n";
17:   return 0;
18: }
```

Tulostus

```
Counter: 11
```

Analyysi

Rivillä 10 alustetaan while-silmukka ehdolla, joka on aina tosi. Silmukka kasvattaa counter-muuttujaa rivillä 12. Rivillä 13 testataan, onko counter ylittänyt arvon 10. Jos näin ei ole, while-silmukka jatkuu. Jos counter on suurempi kuin 10, päättää rivin 14 break-lause silmukan ja ohjelman suoritus palaa riville 16, jossa tulostetaan tulokset.

Ohjelma toimii, mutta se ei ole kovinkaan hieno. Se on hyvä esimerkki väärän työkalun käyttämisestä jonkin työn tekemisessä. Sama asia voitaisiin toteuttaa sijoittamalla counter-arvon testaus sinne, minne se kuuluu eli while-ehtoon.

Varoitus! Ikuiset silmukat kuten while(1) voivat aiheuttaa koneen kaatumisen, mikäli ehtoa ei koskaan saavuteta. Käytä näitä mahdollisuuksia varoen ja testaa ne läpikotaisin.

C++ tarjoaa monia keinoja saman tehtävän toteuttamiseen. Eräs ohjelmoinnin pääideoista on hakea oikea työkalu kunkin tehtävän suorittamiseen.

Tee/Älä tee

Älä käytä goto-lauseita.
Käytä while-silmukoita iterointiin ehdon ollessa tosi.
Käytä continue- ja break-lauseita varoen.
Varmista, että silmukkasi päättyvät joskus.

Do…while -silmukat

On mahdollista, että while-silmukan runkoa ei koskaan suoriteta. While-silmukassa testataan ehto ennen silmukan rungossa olevien ohjelmalauseiden suorittamista ja jos ehto on epätosi heti alussa, ei ohjelmalauseita suoriteta koskaan. Listaus 8.6 havainnollistaa tätä.

Listaus 8.6. while-silmukan rungon ohittaminen.

```
1:  // Listaus 8.6
2:  // Hypätään while-runko, kun
3:  // ehto on epätosi.
4:
5:  #include <iostream.h>
6:
7:  int main()
8:  {
9:    int counter;
10:   cout << "How many hellos?: ";
11:   cin >> counter;
12:   while (counter > 0)
13:   {
14:     cout << "Hello!\n";
15:     counter-- ;
16:   }
17:   cout << "counter is OutPut: " << counter;
18:   return 0;
19: }
```

Tulostus

```
How many hellos?: 2
Hello!
Hello!
Counter is OutPut:0

How many hellos?: 0
Counter is 0
```

Analyysi

Käyttäjää pyydetään antamaan alkuarvo rivillä 10. Se tallennetaan muuttujaan counter. Muuttujan arvo testataan rivillä 12 ja sitä vähennetään while-silmukan rungossa. Kun ensimmäisessä ajossa annettiin arvo 2, suoritettiin while-silmukka kahdesti. Toisella kerralla annettiin counter-muuttujaan arvo 0 ja koska while-silmukan ehto on nyt epätosi, ohitettiin koko while-silmukan runko eikä hello-viestiä tulostettu.

Entä, jos haluaisit varmistaa, että Hello! tulostetaan ainakin kerran? Sitä ei voida toteuttaa while-silmukalla, koska ehto testataan ennen silmukan suorittamista. Voisimme pakottaa tulostuksen if-lauseella seuraavasti:

```
if (counter < 1)   // pakotetaan 1 minimiarvoksi
  counter = 1;
```

Ratkaisu on kuitenkin ruma ja epähieno.

Sen sijaan voisimme käyttää do…while -rakennetta, jossa ehto testataan rakenteen lopussa ja tällöin silmukan runko suoritetaan aina vähintään kerran. Listaus 8.7 esittelee uudelleen kirjoitettua listausta 8.6, jossa nyt käytetään do…while -silmukkarakennetta.

Listaus 8.7. do…while -silmukka.

```
1:  // Listaus 8.7
2:  // do while
3:
4:  #include <iostream.h>
5:
6:  int main()
7:  {
8:    int counter;
9:    cout << "How many hellos? ";
10:   cin >> counter;
11:   do
12:   {
13:     cout << "Hello\n";
14:     counter--;
15:   }  while (counter >0 );
16:   cout << "counter is: " << counter << endl;
17:   return 0;
18: }
```

Tulostus

```
How many hellos? 2
Hello
Hello
Counter is: 0
```

Analyysi

Käyttäjää pyydetään antamaan alkuarvo rivillä 9. Se sijoitetaan muuttujaan counter. Do…while -silmukan runko suoritetaan ennen ehdon testaamista ja siksi runko suoritetaan aina vähintään kerran. Rivillä 13 tulostetaan viesti ja rivillä 14 vähennetään counter-arvoa yhdellä. Rivillä 15 testataan ehto. Jos ehto on tosi, suoritus palaa silmukan alkuun riville 13, muutoin poistutaan silmukasta.

Continue- ja break-lauseet toimivat do…while -silmukassa aivan samoin kuin while-silmukassa.

For-silmukka

While-silmukoiden yhteydessä käytetään ehtoja ja ehdoissa mukanaolevaa muuttujaa käsitellään silmukan rungossa. Listaus 8.8 kuvaa tätä.

Listaus 8.8. Edelleenkin while-silmukan tutkintaa.

```
1:  // Listaus 8.8
2:  // while
3:
4:  #include <iostream.h>
5:
6:  int main()
7:  {
8:    int counter = 0;
9:
10:   while(counter < 5)
11:   {
12:     counter++;
13:     cout << "Looping!  ";
14:   }
15:
16:   cout << "\nCounter: " << counter << ".\n";
17:   return 0;
18: }
```

Tulostus

```
Looping! Looping! Looping! Looping! Looping!
Counter: 5.
```

Analyysi

Ehto asetetaan rivillä 8, jossa counter-muuttujan alkuarvoksi sijoitetaan 0. Rivillä 10 testataan, onko counter-arvo pienempi kuin 5. Rivillä 12 kasvatetaan counter-arvoa. Rivillä 16 tulostetaan suppea viesti. Viestin paikalla voi tietenkin olla paljon tärkeämpiä toimintoja.

For-silmukassa yhdistetään kolme vaihetta yhteen lauseeseen: alustus, testaus ja laskurin kasvatus. For-lause sisältää varatun sanan for, jonka jälkeen tulevat sulkumerkit. Sulkumerkkien sisällä ovat nuo kolme edellä mainittua lausetta erotettuina puolipistein.

Ensimmäinen lause on alustus. Alustuslauseena voi olla mikä tahansa C++ -lause, mutta yleensä siinä vain määritellään ja alustetaan laskurimuuttuja. Toinen lause on testaus, jossa voidaan myöskin käyttää mitä tahansa C++ -ilmauksia. Tämä lause vastaa while-silmukoiden ehtoa. Kolmas lause on toiminto eli yleensä laskuriarvon kasvatus tai vähentäminen. Siinäkin voidaan käyttää mitä tahansa C++ -ohjelmalauseita. Huomaa, että ensimmäisessä ja kolmannessa lauseessa voi olla mitä tahansa C++ -ohjelmalauseita, mutta toisena oleva lause voi olla vain ilmaus eli lause, joka palauttaa arvon. Listaus 8.9 esittelee for-silmukan käyttöä.

Listaus 8.9. for-silmukka.

```
1:  // Listaus 8.9
2:  // for
3:
4:  #include <iostream.h>
5:
6:  int main()
7:  {
8:    int counter;
9:    for (counter = 0; counter < 5; counter++)
10:     cout << "Looping! ";
11:
12:   cout << "\nCounter: " << counter << ".\n";
13:   return 0;
14: }
```

Tulostus

```
Looping! Looping! Looping! Looping! Looping!
Counter: 5.
```

Analyysi

Rivillä 9 oleva for-lause yhdistää laskurin, counter, alustuksen, ehdon (counter < 5) ja laskurin kasvatuksen. Silmukan runko on rivillä 10. Runko voi tietenkin sisältää myös ohjelmalohkon.

Kehittyneet for-lauseet

For-silmukka on tehokas ja joustava. Silmukan kolme itsenäistä lausetta (alustus, testaus ja toiminta) mahdollistavat itsekin lukuisia muunnelmia.

For-silmukka toimii seuraavien vaiheiden mukaan:

1. Suorittaa alustuksen
2. Tarkistaa ehdon
3. Jos ehto on tosi, suorittaa silmukan toiminto-osan ja rungon

Jokaisella kierroksella suoritetaan sitten vaiheet 2 ja 3.

Usea alustus ja kasvattaminen

Ei ole lainkaan harvinaista, että silmukassa alustetaan useampi kuin yksi muuttuja, testataan yhdistetty looginen lauseke ja suoritetaan useampi kuin yksi toiminto. Alustus- ja toiminto-osat voidaan korvata usealla eri C++ -lauseella, jotka erotetaan toisistaan pilkulla. Listaus 8.10 esittelee kahden muuttujan alustamista ja kasvattamista.

Listaus 8.10. Useita lauseita for-silmukassa.

```
1:  //listaus 8.10
2:  // useita lausekkeita
3:  // for-silmukoissa
4:
5:  #include <iostream.h>
6:
7:  int main()
8:  {
9:    for (int i=0, j=0; i<3; i++, j++)
10:     cout << "i: " << i << " j: " << j << endl;
11:   return 0;
12: }
```

Tulostus

```
i: 0  j: 0
i: 1  j: 1
i: 2  j: 2
```

Analyysi

Rivillä 9 alustetaan muuttujat i ja j arvolla 0. Sitten suoritetaan testi (i<3) ja koska se on tosi, suoritetaan for-silmukan runko. Lopuksi suoritetaan kolmas for-lauseen osa, jossa kasvatetaan muuttujia i ja j.

Kun rivi 10 on suoritettu, tarkistetaan ehto uudelleen ja jos se on tosi, suoritetaan kasvattaminen ja runko. Tätä jatketaan, kunnes ehto on epätosi, jolloin silmukasta poistutaan.

Null-lauseet for-silmukassa

Mikä tahansa tai kaikki for-silmukan laueet voivat olla null-lauseita. Tällöin sijoitetaan lauseen paikalle puolipiste. For-lause, joka toimii while-lauseen lailla, saadaan aikaan jättämällä for-silmukasta pois ensimmäinen ja kolmas lause. Listaus 8.11 havainnollistaa tätä menettelyä.

Listaus 8.11. Null-lauseet for-silmukassa.

```
1:  // Listaus 8.11
2:  // for ja null-lauseet
3:
4:  #include <iostream.h>
5:
6:  int main()
7:  {
8:    int counter = 0;
9:
10:   for( ; counter < 5; )
11:   {
12:     counter++;
13:     cout << "Looping!  ";
14:   }
15:
16:   cout << "\nCounter: " << counter << ".\n";
17:   return 0;
18: }
```

Tulostus

```
Looping! Looping! Looping! Looping! Looping!
Counter: 5.
```

Analyysi

Huomannet, että ohjelman for-silmukka muistuttaa aiemmin esiteltyä while-lausetta. Rivillä 9 alustetaan laskurimuuttuja, counter. Rivillä 10 oleva for-lause ei alusta mitään arvoja, mutta suorittaa kuitenkin testin (counter < 5). For-lauseessa ei ole lainkaan kasvatusosaa, joten silmukka toimii aivan kuin se olisi kirjoitettuna seuraavasti:

```
while (counter < 5)
```

Painotamme edelleen, että C++ tarjoaa useita tapoja suorittaa sama asia. Kukaan kokenut C++ -ohjelmoija ei käyttäisi for-silmukkaa edellä kuvatulla tavalla. Menettely havainnollistaa kuitenkin for-lauseen joustavuutta. On itse asiassa mahdollista luoda for-silmukka ilman kolmea alustusvaihetta käyttäen break- ja continue-lauseita. Listaus 8.12 havainnollistaa tätä.

Listaus 8.12. Tyhjä for-silmukan alustusosa.

```
1:  //Listaus 8.12 havainnollistaa
2:  //tyhjää for-lausetta
3:
4:  #include <iostream.h>
5:
6:  int main()
7:  {
8:    int counter=0;       // alustus
9:    int max;
10:   cout << "How many hellos?";
11:   cin >> max;
12:   for (;;)          // a for loop that doesn't end
13:     {
14:     if (counter < max)       // test
15:     {
16:       cout << "Hello!\n";
17:       counter++;          // increment
18:     }
19:   else
20:     break;
21:   }
22:   return 0;
23: }
```

Tulostus

```
How many hellos?  3
Hello!
Hello!
Hello!
```

Analyysi

Nyt for-silmukka on puhdistettu aivan kokonaan. Alustus, testaus ja toiminto on poistettu silmukasta. Alustus tehdään rivillä 8 ennen for-silmukkaa. Testi tehdään erillisellä if-lauseella rivillä 14. Jos testi on tosi, suoritetaan toiminto - laskurin kasvatus - rivillä 17. Jos testi on epätosi, poistutaan silmukasta rivin 20 break-lauseella.

Vaikka tämä esimerkkiohjelma onkin hieman järjetön, on tilanteita, joissa for(;;) - ja while(1) -silmukoille on käyttöä. Saat nähdä esimerkin soveltuvasta käytöstä myöhemmin switch-lauseiden yhteydessä.

Tyhjät for-silmukat

For-silmukan otsikko-osa tarjoaa niin paljon mahdollisuuksia, ettei runko-osaa joskus tarvita lainkaan. Tällöin on muistettava laittaa tyhjä null-lause (;) silmukan rungon paikalle. Puolipiste voi olla samalla rivillä kuin otsikko-osa, mutta tällöin sen havaitseminen vaikeutuu. Listaus 8.13 käyttää tyhjää runko-osaa.

Listaus 8.13. Tyhjä runko for-silmukassa.

```
1:  //Listaus 8.13
2:  //Tyhjä lause
3:  // for-silmukan runkona
4:
5:  #include <iostream.h>
6:  int main()
7:  {
8:    for (int i = 0; i<5; cout << "i: " << i++ << endl)
9:  ;
10:   return 0;
11: }
```

Tulostus

```
i: 0
i: 1
i: 2
i: 3
```

Analyysi

Rivin 8 for-silmukka sisältää kolme lausetta: ensin alustetaan muuttuja i arvolla 0, sitten testataan, onko i < 5 ja lopuksi tulostetaan i:n arvo ja kasvatetaan sitä.

Runko ei tee mitään, joten sen paikalla on tyhjä lause (;). Kyseessä on hieman huono for-silmukka, koska toiminto-osassa on liikaa tehtäviä. Olisikin parempi kirjoittaa silmukka seuraavasti:

```
8: for (int i = 0; i<5; i++)
9:  cout << "i:  "<< i << endl;
```

Nyt koodi on helppolukuisempaa.

Sisäkkäiset silmukat

Silmukoita voidaan laittaa sisäkkäin, yksi silmukka toisen sisälle. Sisällä oleva silmukka tulee suorittaa kokonaan jokaisen ulomman silmukan kierroksen aikana. Listaus 8.14 havainnollistaa tätä. Ohjelmassa tulostetaan merkkejä matriisimuotoon.

Listaus 8.14. Sisäkkäiset for-silmukat.

```
1:  //Listaus 8.14
2:  //sisäkkäiset for-loopit
3:
4:  #include <iostream.h>
5:
6:  int main()
7:  {
8:    int rows, columns;
9:    char theChar;
10:   cout << "How many rows? ";
11:   cin >> rows;
12:   cout << "How many columns? ";
```

```
13:    cin >> columns;
14:    cout << "What character? ";
15:    cin >> theChar;
16:    for (int i = 0; i<rows; i++)
17:    {
18:      for (int j = 0; j<columns; j++)
19:        cout << theChar;
20:      cout << "\n";
21:    }
22:    return 0;
23: }
```

Tulostus

```
How many rows?  4
How many columns?  12
What character?  x
xxxxxxxxxxxx
xxxxxxxxxxxx
xxxxxxxxxxxx
xxxxxxxxxxxx
```

Analyysi

Käyttäjää pyydetään antamaan rivien ja sarakkeiden määrä sekä tulostettava merkki. Ensimmäinen for-silmukka alustaa laskurin (i) arvolla 0 ja sitten suoritetaan ulomman for-silmukan runko.

Rivillä 18 alustetaan toinen for-silmukka. Sen laskuri (j) alustetaan arvolla 0 ja suoritetaan runko-osa. Rivillä 19 tulostetaan valittu merkki ja suoritus palaa sisemmän for-silmukan otsikko-osaan. Laskurin j arvo testataan ja jos testin tulos on tosi, suoritetaan kasvatus ja runko. Rungon ohjelmalause tulostaa jälleen valitun merkin. Tätä jatkuu, kunnes j:n arvo on sama kuin sarakkeiden määrä.

Kun sisäinen silmukka päättyy (ehto on epätosi), suoritus palaa riville 20, jossa tulostetaan tyhjä rivi. Sen jälkeen siirrytään ulomman silmukan otsikko-osaan, jossa testataan, onko i < rows. Jos ehto on tosi, muuttujaa i kasvatetaan ja silmukan runko suoritetaan.

Tämän jälkeen suoritetaan sisempi silmukka uudelleen kokonaan. Silmukassa tulostetaan jälleen rivi valittuja merkkejä.

Switch-lause

If - ja else...if -lauseet menettävät selkeyttään, kun niitä laitetaan sisäkkäin usealle tasolle. Siksi C++ tarjoaakin käyttöön toisen vaihtoehdon, switch-lauseen, jonka avulla voidaan kätevästi saada aikaan rajaton määrä haarautumia ohjelmaan. Switch-lauseen yleinen syntaksi on seuraavanlainen:

```
switch (ilmaus)
{
case arvo_1: lause;
    break;
case arvo_2: lause;
    break;
…
case arvo_n: lause;
    break;
default:    lause;
}
```

switch-komennon jälkeen tuleva ilmaus voi olla mikä tahansa C++ -lauseke ja ohjelmalauseet mitä tahansa C++ -ohjelmalauseita tai lohkoja. Kun switch-rakenne kohdataan, suoritetaan ensin ilmaus ja ilmauksen arvoa verrataan case-osien arvoihin. Huomaa, ettei tässä yhteydessä voi käyttää vertailuoperaattoreita eikä Boolean-operaatioita.

Jos jokin case-osan arvo vastaa ilmauksen arvoa, suoritetaan vastaavat ohjelmalauseet ja siirrytään alaspäin switch-rakenteessa, kunnes kohdataan break-lause. Jos mikään arvo ei vastaa ilmauksen arvoa, suoritetaan valinnainen default-lause. Jos default-osaa ei ole, eikä vastaavaa arvoa löydy, switch-rakenteesta poistutaan.

Huom! On melkein aina hyvä sijoittaa default-osa switch-rakenteeseen. Jos default-osalle ei ole muuta tarvetta, sitä voidaan käyttää testaukseen havaitsemaan mahdottomat case-arvot ja tulostamaan vastaavat viestit. Siitä voi olla merkittävää apua vianhaussa.

On tärkeää huomata, että jos rakenteessa ei ole break-osaa case-osan jäljessä, siirrytään seuraavaan case-osaan. Tällainen menettely on joskus tarpeellista, mutta usein se on virhe. Jos päätät sallia suorituksen siirtyvän alaspäin, muista laittaa koodiin kommentti kertomaan, ettei kyseessä ole virhe.

Listaus 8.15 havainnollistaa switch-rakenteen käyttöä.

Listaus 8.15. switch-rakenteen esittely.

```
1:  //Listaus 8.15
2:  // switch
3:
4:  #include <iostream.h>
5:
6:  int main()
7:  {
8:    unsigned short int number;
9:    cout << "Enter a number between 1 and 5: ";
10:   cin >> number;
11:   switch (number)
12:   {
13:     case 0:   cout << "Too small, sorry!";
14:     break;
```

```
15:     case 5:  cout << "Good job!\n";  // läpi
16:     case 4:  cout << "Nice Pick!\n"; // läpi
17:     case 3:  cout << "Excellent!\n"; // läpi
18:     case 2:  cout << "Masterful!\n"; // läpi
19:     case 1:  cout << "Incredible!\n";
20:     break;
21:     default: cout << "Too large!\n";
22:     break;
23:   }
24:   cout << "\n\n";
25:   return 0;
26: }
```

Tulostus

```
Enter a number between 1 and 5:  3
Excellent!
Masterful!
Incredible!

Enter a number between 1 and 5:  8
Too large!
```

Analyysi

Käyttäjää pyydetään antamaan luku, jonka switch-lause ottaa vastaan. Jos numero on 0, rivin 13 case-lause vastaa sitä ja tulostetaan viesti "Too small, sorry!" ja break-lause päättää switch-lauseen. Jos arvo on 5, siirrytään riville 15, jossa tulostetaan viesti. Sen jälkeen siirrytään riville 16, jossa tulostetaan jälleen viesti. Tätä jatketaan, kunnes kohdataan rivin 20 break-lause.

Ohjelman ydin on siinä, että annettaessa luku väliltä 1-5 tulostetaan useita viestejä. Jos arvo ei ole väliltä 0-5, odotetaan sen olevan liian suuri, jolloin suoritetaan default-osa rivillä 21.

Yhteenveto

C++ -ohjelmissa voidaan toteuttaa haarautuminen monin eri tavoin. While-silmukat tarkistavat ehdon ja jos se on tosi, suoritetaan silmukan rungon ohjelmalauseet. Do…while -rakenteen runko suoritetaan ensin, jonka jälkeen tarkistetaan ehto-osa. For-silmukoiden otsikossa on arvon alustus ja testi. Jos testi on tosi, suoritetaan otsikon toiminto-osa sekä silmukan runko. Joka kierroksella testi suoritetaan uudelleen.

Goto-lausetta tulisi välttää, koska se aiheuttaa ehdottoman hypyn epämääräiseen ohjelman kohtaan, jolloin ohjelmaa on vaikea lukea, ymmärtää ja ylläpitää. Continue-lauseella voidaan siirtyä while-, do…while- ja for-silmukan alkuun ja break aiheuttaa silmukoista poistumisen.

Kysymyksiä ja Vastauksia

K
Kuinka valitsen if…else- ja switch-lauseen välillä?
V
Jos ohjelmassa on enemmän kuin yksi tai kaksi else-lausetta ja testaat samaa arvoa, harkitse switch-lauseen käyttöä.
K
Kuinka valitsen while- ja do…while -lauseiden välillä?
V
Jos silmukan runko tulee suorittaa vähintään kerran, harkitse do…while -silmukan käyttöä; yritä käyttää while-lausetta muulloin.
K
Kuinka valitsen while- ja for-silmukoiden välillä?
V
Jos alustat laskurimuuttujan, testaat ehdon ja kasvatat laskuria jokaisella silmukkakierroksella, on for-silmukka mahdollisesti sopivampi. Jos muuttujasi on jo alustettu etkä kasvata sitä joka kierroksella, soveltunee while paremmin käyttöön.
K
Kumpi on parempi: while(1) vai for(;;)?
V
Lauseilla ei ole mitään merkittävää eroa.