Osa

I

4. oppitunti

Ilmaukset ja ohjelmalauseet

Ohjelma ei todellisuudessa ole mitään muuta kuin joukko komentoja, jotka suoritetaan peräkkäin. Monipuolisuutta ohjelmaan saadaan, kun ohjelmassa haaraudutaan suorittamaan komentoja toisaalla sen mukaan, onko tietty ohjelmalause tosi vai epätosi. Tässä luvussa opit seuraavia asioita:

- ❏ Mitä ohjelmalauseet ovat
- ❏ Mitä ovat ilmaukset
- ❏ Kuinka operaattoreita käytetään
- ❏ Totuusarvojen merkitys

Ohjelmalauseet

Uusi käsite Kaikki C++ -lauseet päättyvät puolipisteeseen. Ohjelmalause voi valvoa suoritusta, aikaansaada suorituksen tai sitten se ei tee mitään (null-lause).

Sijoitus on yksinkertainen ohjelmalause:

```
x = a + b;
```

Toisin kuin algebrassa, tämä lause ei tarkoita, että x on yhtä suuri kuin a + b. Lause luetaan seuraavasti: "Laske a:n ja b:n summa ja sijoita se x:ään". Vaikka lause tekeekin kaksi toimintoa, se on yksi lause ja päättyy puolipisteeseen. Sijoitusoperaattori lisää oikealla puolella olevan tiedon vasemmalla puolella olevaan muuttujaan.

Tulostumattomat merkit

Uusi käsite Välilyönnit, sarkainlyönnit ja rivinvaihdot ovat tulostumattomia merkkejä. Kääntäjä ohittaa ne. Välilyönnin sijaan voidaan aina sijoittaa sarkainpainallus tai rivinvaihto. Välit tekevät ohjelmasta luettavamman.

Edellä kirjoitettu sijoituslause voitaisiin kirjoittaa myös näin:

```
x=a+b;
```

tai

```
x       =a
+   b   ;
```

Vaikkakin viimeinen lause on täysin laillinen, se näyttää järjettömältä. Väleillä lisätään luettavuutta koodiin eikä sillä tule heikentää sitä. C++ tarjoaa ohjelmointiresurssit, mutta sinulla tulee olla maalaisjärkeä sen käytössä.

Yhdistetyt lauseet

Uusi käsite Kaikkialle, minne voit sijoittaa yksittäisen lauseen, voit sijoittaa myös yhdistetyn lauseen. Yhdistetty lause alkaa alkavalla aaltosululla ({) ja

päättyy sulkevaan aaltosulkuun (}).

Vaikka jokaisen yhdistetyn lauseen ohjelmalauseen tulee päättyä puolipisteeseen, ei itse yhdistetty lause pääty puolipisteeseen. Esimerkiksi:

```
{
    temp = a;
    a = b;
    b = temp;
}
```

Edellä olevat lauseet aikaansaavat muuttujien a ja b arvojen vaihtumisen.

Tee/Älä tee

Käytä aaltosulkuja pareittain.
Päätä ohjelmalauseet puolipisteeseen.
Käytä välilyöntejä harkiten luettavuuden lisäämiseen.

Ilmaukset

Uusi käsite

Ilmaus palauttaa aina arvon.

Siinä se on, selvästi ja yksinkertaisesti. Jos lause palauttaa arvon, se on ilmaus. Kaikki ilmaukset ovat myös ohjelmalauseita.

Seuraavassa on esimerkkejä erilaisista ilmauksista:

```
3.2                 // palauttaa arvon 3.2
Pi                  // palauttaa vakioarvon 3.14
SecondsPerMinute    // kokonaislukuvakio, joka palauttaa 60
```

Olettaen, että PI on vakio (3.14) ja SecondsPerMinute vakioarvo 60, ovat kaikki nuo edellä olevat lauseet ilmauksia.

Monimutkainen ilmaus

```
x = a + b;
```

ei pelkästään lisää a:n ja b:n summaa muuttujaan x vaan myöskin palauttaa x:n arvon. Niinpä tuokin lause on ilmaus. Koska kyseessä on ilmaus, voidaan lause sijoittaa sijoituslauseen oikealle puolelle:

```
y = x = a + b;
```

Tuo rivi suoritetaan nyt seuraavassa järjestyksessä:

Lisää a b:hen
Sijoita summa x:ään
Sijoita x muuttujaan y

Jos muuttujat a, b, x ja y ovat kokonaislukumuuttujia ja a saa arvon 2 sekä b arvon 5, tulee sekä x:n että y:n arvoksi 7.

Listaus 4.1 havainnollistaa monimutkaisten ilmausten suorittamista.

Listaus 4.1. Monimutkaisia ilmauksia.

```
13: #include <iostream.h>
14: int main()
15: {
16:   int a=0, b=0, x=0, y=35;
17:   cout << "a: " << a << " b: " << b;
18:   cout << " x: " << x << " y: " << y << endl;
19:   a = 9;
20:   b = 7;
21:   y = x = a+b;
22:   cout << "a: " << a << " b: " << b;
23:   cout << " x: " << x << " y: " << y << endl;
24:   return 0;
25: }
```

Tulostus

```
a: 0 b: 0 x: 0 y: 35
a: 9 b: 7 x: 16 y: 16
```

Analyysi

Rivillä 4 määritellään ja alustetaan neljä muuttujaa ja niiden arvot tulostetaan riveillä 5 ja 6. Rivillä 7 sijoitetaan a:han arvo 9. Rivillä 8 sijoitetaan b:hen arvo 7. Rivillä 9 lasketaan a:n ja b:n arvot yhteen ja sijoitetaan summa x:ään. Tämä ilmaus (x = a + b) suoritetaan ja arvo sijoitetaan muuttujaan y.

Operaattorit

Uusi käsite Operaattori on symboli, joka saa kääntäjän suorittamaan toiminnon.

Sijoitusoperaattori

Uusi käsite Sijoitusoperaattori (=) sijoittaa oikealla puolella olevan ilmauksen arvon vasemmalla puolella olevaan muuttujaan. Ilmaus

```
x = a + b;
```

sijoittaa a:n ja b:n summan muuttujaan x.

Sijoitusoperaattorin vasemmalla puolella olevaa operandia kutsutaan nimellä lvalue ja oikealla puolella olevaa arvoa taas nimellä rvalue.

Vakiot voivat olla vain oikealla puolella kuten seuraavassa lauseessa:

```
x = 35; // ok
```

Et voi kuitenkaan kirjoittaa

```
35 = x; // virhe, väärä lvalue!
```

Uusi käsite Lvalue on operandi, joka voi olla ilmauksen vasemmalla puolella. Rvalue voi olla ilmauksen oikealla puolella.

Matemaattiset operaattorit

Käytössä on viisi matemaattista operaattoria: yhteenlasku (+), vähennyslasku (-), kertolasku (*), jakolasku (/) ja jakojäännös (%).

Yhteen-, vähennys- ja kertolasku ovat helppotajuisia ja toimivat odotetusti. Jakolaskuoperaattorin kanssa on hieman toisin.

Kokonaislukujakolasku poikkeaa normaalista jakolaskusta. Kun jaat luvun 21 luvulla 4, on tuloksena reaaliluku (desimaaliluku). Kokonaisluvuilla ei kuitenkaan ole murto-osia, joten murto-osat heitetään ulos ja tulokseksi tulee 5.

Jakojäännös-operaattori palauttaa kokonaislukujakolaskun jakojäännöksen. Niinpä operaatio 21 % 4 antaisi tulokseksi 1, koska 21 / 4 on 5 (ja jäljelle jää 1).

Jakojäännöksen löytäminen voi olla hyödyllistä. Voit esimerkiksi haluta tulostaa joka kymmenennen ohjelmalauseen. Jos numero on kymmenen kerrannainen, antaa jakojäännösoperaattori tulokseksi 0, mitä tietoa voi sitten hyödyntää tulostuksessa.

Sijoitus- ja matemaattisten operaattoreiden yhdistäminen

On melko yleistä lisätä muuttujaan jokin arvo ja sijoittaa sitten tulos takaisin muuttujaan. Jos muuttujan nimeltä myAge arvoa halutaan kasvattaa kahdella, voidaan kirjoittaa:

```
int myAge = 5;
int temp;
```

```
temp = myAge + 2;   //  summaa  2  ja  5  ja  sijoita  summa
muuttujaan temp
myAge = temp;   // sijoita tulos takaisin muuttujaan myAge
```

Yllämainittu menettely on kuitenkin hyvin tehoton ja tuhlaavainen. C++ -kielessä voidaan sijoittaa sama muuttuja molemmille puolille sijoitusoperaattoria eli voidaan kirjoittaa:

```
myAge = myAge + 2;
```

Tuo koodi on hyvin tehokasta. Matematiikassa tuo ilmaus olisi merkityksetön, mutta C++ -kielessä se tarkoittaa "Lisää luku 2 muuttujaan myAge ja sijoita tulos muuttujaan myAge".

On mahdollista kirjoittaa vieläkin yksinkertaisemmin:

```
myAge += 2;
```

Tuo koodi ei ole niin helppolukuista. Yhdistetty operaattori (+=) lisää oikealla puolella olevan arvon lvalue-arvoon ja sijoittaa sitten tuloksen lvalue-arvoon. Jos muuttujan myAge arvo olisi aluksi 4, olisi sen arvona operaation jälkeen 6. Muita yhdistettyjä operaattoreita ovat vähentäminen (-=), jakaminen (/=), kertominen (*=) ja jakojäännös (%=).

Inkrementointi (kasvatus) ja dekrementointi (vähentäminen)

Uusi käsite Yleisin muuttujaan lisättävä tai siitä vähennettävä arvo on 1. C++ -kielessä kutsutaan ykkösen lisäämistä inkrementoinniksi ja ykkösen vähentämistä dekrementoinniksi. Näitä toimenpiteitä varten on omat operaattorinsa.

Inkrementtioperaattori (++) kasvattaa muuttujan arvoa yhdellä ja dekrementtioperaattori (--) taas vähentää muuttujan arvoa yhdellä. Jos käytössä on muuttuja C, voidaan sen arvoa kasvattaa yhdellä seuraavasti:

```
C++;    // aloittaa C:llä ja inkrementoi sitä
```

Tuo lause on yhdenmukainen kuvaavamman lauseen kanssa:

```
C = C + 1;
```

Ja edellä oleva lause on yhdenmukainen seuraavan lauseen kanssa:

```
C += 1;
```

Etu- ja jälkiliiteoperaattorit

Uusi käsite Sekä inkrementti (++) että dekrementtioperaattori (--) voivat sijaita joko muuttujan edessä (etuliite) tai jäljessä (jälkiliite). Jos vaikkapa inkrementtioperaattori on muuttujan edessä, kirjoitetaan ++myAge ja jos se on jäljessä myAge++.

Yksinkertaisessa lauseessa ei ole suurta merkitystä sillä, kummalla puolella operaattori on. Sen sijaan monimutkaisessa lauseessa, jossa inkrementoidun tai dekrementoidun muuttujan arvo sijoitetaan toiseen muuttujaan, on sijainnilla merkitystä. Etuliiteoperaattori suoritetaan ennen sijoitusta ja jälkiliiteoperaattori taas sijoituksen jälkeen.

Etuliite toimii näin: kasvatetaan ensin arvoa ja otetaan arvo esille vasta sen jälkeen. Jälkiliitteen kohdalla haetaan ensin arvo ja kasvatetaan sitten alkuperäistä arvoa.
Tuo voi kuulostaa aluksi hieman sekavalta, mutta katsotaan esimerkkiä. Jos muuttujan x alkuarvo on 5 ja kirjoitetaan

```
int a = ++x;
```

on x:ää kasvatettu arvoon 6 ja sijoitettu sitten tuo arvo muuttujaan a. Nyt a:n arvo on 6 ja x:n arvo myös 6.

Jos tämän jälkeen kirjoitat

```
int b = x++;
```

sijoitetaan ensin arvo 6 muuttujaan b ja sen jälkeen kasvatetaan x:n arvoa yhdellä. Nyt b:n arvo on siis 6 ja x:n arvo 7. Listaus 4.2 esittelee molempia menettelytapoja.

Listaus 4.2. Etu- ja jälkiliiteoperaattorit.

```
1: // Listaus 4.2 - esittelee etu- ja
2: // jälkiliiteoperaattoreita: inkre-
3: // mentointi ja dekrementointi
4: #include <iostream.h>
5: int main()
6: {
7:   int myAge = 39;      // int-alustus
8:   int yourAge = 39;
9:   cout << "I am:\t" << myAge << "\tyears old.\n";
10:  cout << "You are:\t" << yourAge << "\tyears old\n";
11:  myAge++;         // inkrementointi jäljessä
12:  ++yourAge;       // inkrementointi edessä
13:  cout << "One year passes...\n";
14:  cout << "I am:\t" << myAge << "\tyears old.\n";
15:  cout << "You are:\t" << yourAge << "\tyears old\n";
16:  cout << "Another year passes\n";
17:  cout << "I am:\t" << myAge++ << "\tyears old.\n";
18:  cout << "You are:\t" << ++yourAge << "\tyears old\n";
```

```
19:   cout << "Let's print it again.\n";
20:   cout << "I am:\t" << myAge << "\tyears old.\n";
21:   cout << "You are:\t" << yourAge << "\tyears old\n";
22:   return 0;
23: }
```

Tulostus

```
I am:   39 years old.
You are:    39 years old.
One year passes…
I am:   40 years old.
You are:    40 years old.
Another year passes
I am:   40 years old.
You are:    41 years old.
Let's print it again.
I am:   41 years old.
You are:    41 years old.
```

Analyysi

Riveillä 7 ja 8 esitellään kaksi muuttujaa ja alustetaan ne arvolla 39. Niiden arvot tulostetaan riveillä 9 ja 10.

Rivillä 11 inkrementoidaan myAge-muuttujaa ja inkrementtioperaattori on etuliitteenä. Rivillä 12 inkrementoidaan yourAge-muuttujaa operaattorin ollessa jälkiliitteenä. Tulokset tulostetaan riveillä 14 ja 15 (tulokset ovat samoja (40)).

Rivillä 17 inkrementoidaan myAge-muuttujaa sen ollessa osana tulostuslausetta ja inkrementointioperaattori on muuttujan jäljessä. Tällöin kasvattaminen tapahtuu tulostamisen jälkeen, joten arvo 40 tulostetaan. Sen sijaan kasvatetaan rivillä 18 yourAge-muuttujaa inkrementtioperaattorin ollessa muuttujan edessä. Tällöin muuttujan arvoa kasvatetaan ennen tulostamista ja tulostettava arvo on siten 41.

Lopuksi tulostetaan riveillä 20 ja 21 arvot uudelleen. Koska inkrementointi on toteutettu, tulostuu nyt arvo 41 molempien muuttujien kohdalla.

Suoritusjärjestys

Mikä operaatio suoritetaan ensin esimerkiksi seuraavassa monimutkaisemmassa lauseessa?

```
x = 5 + 3 * 8;
```

Jos yhteenlasku suoritetaan ensin, saa x arvon 64. Jos kertolasku suoritetaan ensin, saa x arvon 29.

Uusi käsite

Jokaisella operaattorilla on prioriteettiarvonsa. Täydellinen listaus on liitteessä A, "Operaattoreiden suoritusjärjestys". Kertomisella on suurempi prioriteetti kuin yhteenlaskulla, joten edellä olevan lauseen tulokseksi tulee 29.

Kun kahdella matemaattisella operaattorilla on sama prioriteetti, ne suoritetaan vasemmalta oikealle. Täten lauseessa

```
x = 5 + 3 + 8 * 9 + 6 * 4;
```

suoritetaan ensin kertolaskut vasemmalta oikealle ja sen jälkeen yhteenlaskut vasemmalta oikealle. Ensimmäisen vaiheen tuloksena olisi lause seuraavassa muodossa

```
x = 5 + 3 + 72 + 24;
```

Sitten suoritetaan yhteenlaskut ja saadaan tulokseksi 104.

Ole huolellinen. Jotkut operaattorit kuten sijoitusoperaattori suoritetaan järjestyksessä oikealta vasemmalle! Entäpä, jos suoritusjärjestys ei täytä tarpeitasi? Ajattele vaikkapa seuraavaa lausetta

```
TotalSeconds = NumMinutesToThink + NumMinutesToType * 60;
```

Nyt ei halutakaan kertoa muuttujaa NumMinutesToType luvulla 60 ja lisätä tuloa muuttujaan NumMinutesToThink. Sen sijaan tavoitteena on ensin laskea yhteen minuuttimäärät eli muuttujat NumMinutesToThink ja NumMinutesToType ja kertoa sitten tuo summa luvulla 60, jotta esille saadaan sekuntien kokonaismäärä.

Tässä tapauksessa voidaan käyttää sulkumerkkejä muuttamaan suoritusjärjestystä. Sulkumerkkien sisällä olevilla lausekkeilla on suurempi prioriteetti kuin millään muulla matemaattisella operaattorilla. Täten lause

```
TotalSeconds = (NumMinutesToThink + NumMinutesToType) * 60;
```

antaisi oikean tuloksen.

Sisäkkäiset sulkumerkit

Monimutkaisissa lausekkeissa voidaan joutua sijoittamaan sulkumerkkejä sisäkkäin. Jos on esimerkiksi laskettava sekuntien kokonaismäärä ja sen jälkeen saatava esille mukana olevien henkilöiden määrä ja lopuksi kerrottava henkilöiden määrä sekuntien määrällä:

```
TotalPersonSeconds = ( ( (NumMinutesToThink + NumMinutesToType
* 60) * (PeopleInTheOffice + PeopleOnVacation) )
```

Tällainen lauseke luetaan sisältä ulospäin. Ensin summataan NumMinutesToType ja NumMinutesToThink, koska ne ovat sisimpien sulkujen sisällä. Sitten tuo summa kerrotaan luvulla 60. Seuraavaksi summataan PeopleInTheOffice ja PeopleOnVacation ja lopuksi kerrotaan tuo summa sekuntien kokonaismäärällä.

Tuo lauseke on hieman vaikealukuinen ja hankala muokattava. Nyt onkin tärkeää pohtia, voisimmeko kirjoittaa lauseen selkeämmin. Seuraavassa lause on kirjoitettu uudelleen käyttäen apuna tilapäismuuttujia:

```
TotalMinutes = NumMinutesToThink + NumMinutesToType;
TotalSeconds = TotalMinutes * 60;
TotalPeople = PeopleInTheOffice + PeopleOnVacation;
TotalPersonSeconds = TotalPeople * TotalSeconds;
```

Esimerkin kirjoittaminen kestää kauemmin ja siinä käytetään enemmän tilapäisiä muuttujia kuin edellä, mutta koodi on kuitenkin ymmärrettävämpää. Lisää vielä kommentti alkuun ja kerro siinä, mitä koodi tekee. Samoin voit muuttaa luvun 60 symboliseksi vakioksi, joka kuvaa sekuntien määrää minuutissa. Sen jälkeen koodi on helppolukuista ja helpommin ylläpidettävää.

Totuusarvot

C++ -kielessä pidetään arvoa nolla (0) arvona epätosi (false) ja kaikkia muita arvoja arvona tosi (true). Jos ilmauksen tulos on siis epätosi, on ilmauksen arvona 0.

Vertailuoperaattorit

Vertailuoperaattoreita käytetään määrittämään, ovatko arvot yhtäsuuria tai onko toinen arvo suurempi tai pienempi kuin toinen. Jokainen vertailuoperaattori palauttaa joko arvon 1 (tosi) tai 0 (epätosi). Vertailuoperaattorit on esitetty taulukossa 4.1.

Jos kokonaislukumuuttujalla myAge on arvo 39 ja muuttujalla yourAge taas arvo 40, voidaan niiden yhtäsuuruus testata yhtäsuuruusoperaattorilla (==):

```
myAge == yourAge;   // onko myAge yhtäsuuri kuin yourAge?
```

Ilmauksen arvo on nolla, epätosi, koska muuttujat ovat erisuuria. Ilmaus

```
myAge > yourAge;  // onko myAge suurempi kuin yourAge?
```

antaa arvoksi nollan eli epätosi.

Varoitus! Monet aloittelevat C++ -ohjelmoijat sotkevat joskus sijoitusoperaattorin (=) ja yhtäsuuruusoperaattorin (==), mikä voi aiheuttaa pahoja virheitä ohjelmaan.

Vertailuoperaattoreita on kuusi erilaista: yhtä suuri (==), pienempi kuin (<), suurempi kuin (>), pienempi tai yhtä suuri kuin (>=) ja suurempi tai yhtä suuri kuin (>=) ja eri suuri kuin (!=). Taulukossa 4.1 on esitetty kukin vertailuoperaattori sekä esimerkki sen käytöstä.

Taulukko 4.1. Vertailuoperaattorit

Nimi	Operaattori	Näyte	Tulos
Yhtä suuri	==	100==50;	epätosi
		50 == 50;	tosi
Eri suuri	!=	100!=50;	tosi
		50 != 50;	epätosi
Suurempi kuin	>	100>50;	tosi
		50 > 50;	epätosi
Suurempi tai yhtä suuri kuin	>=	100>=50	tosi
		50 >= 50;	epätosi
Pienempi kuin	<	100<50;	epätosi
		50 < 50;	epätosi
Pienempi tai yhtä suuri kuin	<=	100<=50;	epätosi
		50 <= 50;	tosi

If-lause

Normaalisti ohjelman kulku on suoraviivainen ja lähdekoodia suoritetaan rivi riviltä. If-lause mahdollistaa ehdon testaamisen (esimerkiksi yhtäsuuruustestin) ja haarautumisen ohjelmassa ehdon palauttaman arvon mukaan.

If-lauseen yksinkertaisin muoto on seuraavanlainen:

```
if (ilmaus)
    Ohjelmalauseet;
```

Sulkumerkeissä oleva ilmaus voi olla mikä tahansa lauseke, mutta yleensä se on vertailulauseke. Jos ilmaus saa arvon nolla, on tulos epätosi ja

ohjelmalauseet ohitetaan. Jos ilmaus saa nollasta poikkeavan arvon, on tulos tosi ja ohjelmalausekkeet suoritetaan. Katso seuraavaa esimerkkiä:

```
if (bigNumber > smallNumber)
    bigNumber = smallNumber;
```

Koodi vertaa muuttujia bigNumber ja smallNumber. Jos bigNumber on suurempi, sijoittaa alempi lause siihen arvon smallNumber.

Else-lause

Ohjelmassa voidaan haarautua suorittamaan jotain koodia ehdon ollessa tosi ja jotain muuta koodia ehdon ollessa epätosi. Tämä voidaan toteuttaa useammalla allekkaisella if-lauseella, mutta parempi vaihtoehto on käyttää else-lausetta:

```
if (ilmaus)
    Ohjelmalauseet;
else
    Ohjelmalauseet;
```

Listaus 4.3 esittelee if-else -rakenteen käyttöä.

Listaus 4.3. if-else -rakenne.

```
1:  // Listaus 4.3 - esittelee if-lausetta
2:  // else-osan kanssa
3:  #include <iostream.h>
4:  int main()
5:  {
6:    int firstNumber, secondNumber;
7:    cout << "Please enter a big number: ";
8:    cin >> firstNumber;
9:    cout << "\nPlease enter a smaller number: ";
10:   cin >> secondNumber;
11:   if (firstNumber > secondNumber)
12:     cout << "\nThanks!\n";
13:   else
14:     cout << "\nOops. The second is bigger!";
15:
16:   return 0;
17: }
```

Tulostus

```
Please enter a big number: 10
Please enter a smaller number: 12
Oops. The second is bigger!
```

Analyysi

Rivillä 11 on if-lause. Jos ehto on tosi, suoritetaan rivin 12 koodi; jos ehto on epätosi, suoritetaan rivin 14 koodi. Jos rivin 13 else-lause poistettaisiin, suoritettaisiin rivin 14 koodi riippumatta siitä, onko if-lauseen ehto tosi vai

epätosi. Muista, että if-lause päättyy rivin 12 jälkeen. Jos else-osaa ei olisi, olisi rivi 14 pelkästään erillinen, suoritettava ohjelmarivi.

Jompikumpi tai molemmat ohjelmalauseet voitaisiin korvata koodilohkolla, joka on aaltosuluissa.

if-lause

```
if (ilmaus)
    Ohjelmalause;
Seuraava ohjelmalause;
```

Jos ilmaus on tosi, suoritetaan ohjelmalause ja ohjelma jatkaa seuraavalla ohjelmalauseella. Jos ilmaus ei ole tosi, ohjelmalause ohitetaan ja suoritetaan seuraava ohjelmalause.

Ohjelmalause voi olla yksittäinen ohjelmalause, joka päättyy puolipisteeseen, tai aaltosuluissa oleva koodilohko.

Monipuoliset if-lauseet

On hyvä huomata, että if- ja else-lauseessa voidaan käyttää mitä tahansa ohjelmalausetta vaikkapa toista if- tai else-lausetta. Ohjelmissa voidaan käyttää vaikkapa seuraavanlaisia muotoja:

```
if (ilmaus1)
{
    if (ilmaus2)
        ohjelmalause1;
    else
    {
        if (ilmaus3)
            ohjelmalause2;
        else
            ohjelmalause3;
    }
}
else
    ohjelmalause4;
```

Voit lukea lauseen seuraavasti: "Jos ilmaus1 on tosi ja ilmaus2 on tosi, suorita ohjelmalause1. Jos ilmaus1 on tosi, mutta ilmaus2 epätosi ja jos sitten ilmaus3 on tosi, suorita ohjelmalause2. Jos ilmaus1 on tosi, mutta ilmaus2 ja ilmaus3 ovat epätosia, suorita ohjelmalause3. Lopuksi, jos ilmaus1 on epätosi, suorita ohjelmalause4". Kuten näet, ovat monimutkaiset if-lauseet sekavia!

Listauksessa 4.4 on esitetty toinen monimutkainen if-lause.

Listaus 4.4. Sisäkkäiset if-lauseet, monimutkainen rakenne.

```
1:  // Listaus 4.4 - sisäkkäiset
2:  // if-lauseet
3:  #include <iostream.h>
4:  int main()
5:  {
6:  // Pyydetään 2 lukua
7:  // Sijoitetaan arvot muuttujiin bigNumber ja littleNumber
8:  // Jos bigNumber on suurempi kuin littleNumber,
9:  // katsotaan ovatko ne jaettavissa.
10: // Jos ovat, katsotaan, onko arvo sama
11:   int firstNumber, secondNumber;
12:   cout << "Enter two numbers.\nFirst: ";
13:   cin >> firstNumber;
14:   cout << "\nSecond: ";
15:   cin >> secondNumber;
16:   cout << "\n\n";
17:
18:   if (firstNumber >= secondNumber)
19:   {
20:     if ( (firstNumber % secondNumber) == 0) // evenly divisible?
21:     {
22:       if (firstNumber == secondNumber)
23:         cout << "They are the same!\n";
24:       else
25:         cout << "They are evenly divisible!\n";
26: }
27:       else
28:         cout << "They are not evenly divisible!\n";
29: }
30:   else
31:     cout << "Hey! The second one is larger!\n";
32:   return 0;
33: }
```

Tulostus

```
Enter two numbers.
First: 10
Second: 2
They are evenly divisible!
```

Analyysi

Ohjelma kehottaa syöttämään kaksi lukua, joita sitten vertaillaan. Rivin 18 if-lause testaa, onko ensimmäinen numero suurempi tai yhtäsuuri kuin toinen numero. Jos ei ole, suoritetaan rivin 31 else-lause.

Jos ensimmäinen if-lause on tosi, suoritetaan riviltä 21 alkava koodilohko ja testataan toinen, rivin 22 if-lause. Toinen ehto testaa, onko ensimmäisen ja toisen numeron jakojäännös nolla. Jos näin on, ovat numerot tasan jaollisia tai yhtä suuria. Rivin 22 if-lause testaa sitten yhtäsuuruuden ja tulostaa vastaavan viestin.

Jos rivin 22 if-lause on epätosi, suoritetaan rivin 28 else-osa.

Käytä sulkumerkkejä sisäkkäisissä if-lauseissa

Sisäkkäisten if-lauseiden yhteydessä ei ole pakko käyttää sulkumerkkejä:

```
if (x > y)  // jos x on suurempi kuin y
    if (x < z)  // ja jos x on pienempi kuin z
        x = y;  // niin sijoita y:n arvo x:ään
```

Kuitenkin laajempien sisäkkäisten if-lauseiden yhteydessä sulkumerkit lisäävät koodin luettavuutta. Muista myös välilyöntien ja sisennysten merkitys. On helppo menettää ohjelman logiikka ja sotkeentua if- ja else-osien kanssa laajemmissa rakenteissa. Listaus 4.5 havainnollistaa tätä ongelmaa.

Listaus 4.5. if- ja else-osien sekaantuminen ilman sulkumerkkien käyttöä.

```
1:  // Listaus 4.5 - esittelee aaltosulkujen
2:  // käyttöä sisäkkäisissä if-lauseissa
3:  #include <iostream.h>
4:  int main()
5:  {
6:    int x;
7:    cout << "Enter a number less than 10 or greater than 100: ";
8:    cin >> x;
9:    cout << "\n";
10:
11:   if (x > 10)
12:   if (x > 100)
13:     cout << "More than 100, Thanks!\n";
14:   else                          // ei oikea else
15:     cout << "Less than 10, Thanks!\n";
16:
17:   return 0;
18: }
```

Tulostus

```
Enter a number less than 10 or greater than 100: 20
Less than 10, thanks!
```

Analyysi

Ohjelmoijan aikomuksena oli pyytää antamaan luku väliltä 10 ja 100, tarkistaa arvo ja tulostaa sitten kiitosviesti.

Jos rivin 11 if-lause on tosi, suoritetaan seuraava ohjelmalause. Tässä tapauksessa suoritetaan rivi 12, kun luku on suurempi kuin 10. Rivi 12 sisältää myöskin if-lauseen. Tämä if-lause saa arvon tosi, jos annettu luku on suurempi kuin 100. Jos luku ei ole suurempi kuin 100, suoritetaan rivi 13.

Jos annettu numero ei ole pienempi tai yhtäsuuri kuin 10, saa rivin 10 if-lause arvon epätosi. Tällöin ohjelma siirtyy seuraavalle riville (rivi 16). Jos annat luvun, joka on pienempi kuin 10, on tulostus seuraavanlainen:

```
Enter a number less than 10 or greater than 100: 9
```

Rivin 14 else-lause oli selvästikin tarkoitettu rivin 11 if-lauseen yhteyteen. Valitettavasti tuo else-osa kytkeytyykin rivin 12 if-lauseeseen, jolloin tapahtuu virhe.

Kääntäjä ei huomaa virhettä eikä generoi virheilmoitusta. Kyseessä on toimiva C++ -ohjelma, mutta se ei vain toimi oikein. Kaiken lisäksi se toimii testeissä pitkälti oikein, etenkin jos ohjelmalle syötetään arvoja, jotka ovat suurempia kuin 100.

Ohjelma on korjattuna listauksessa 4.6.

Listaus 4.6. Sulkumerkkien oikea käyttö if-lauseissa.

```
1:  // Listaus 4.5 - esittelee aaltosulkujen
2:  // käyttöä
3:  #include <iostream.h>
4:  int main()
5:  {
6:    int x;
7:    cout << "Enter a number less than 10 or greater than 100: ";
8:    cin >> x;
9:    cout << "\n";
10:
11:   if (x > 10)
12:   {
13:     if (x > 100)
14:       cout << "More than 100, Thanks!\n";
15:   }
16:     else                          // ei oikea else
17:       cout << "Less than 10, Thanks!\n";
18:   return 0;
19: }
```

Tulostus

```
Enter a number less than 10 or greater than 100: 20
```

Analyysi

Riveillä 12 ja 15 olevat sulkumerkit aikaansaavat koodilohkon, jolloin rivin 16 else-osa liittyy rivin 11 if-lauseeseen kuten pitikin.

Käyttäjä kirjoitti luvun 20, jolloin rivin 11 if-lause on tosi; rivin 13 if-lause on epätosi, joten mitään ei tulosteta. Olisi parempi, jos ohjelmoija panisi toisen else-lauseen rivin 14 jälkeen, jotta virheet saataisiin esille.

Huom! Tämän kirjan esimerkkiohjelmat on tarkoitettu tiettyjen rakenteiden ja ominaisuuksien esittelyyn. Ne on pidetty tarkoituksellisesti

yksinkertaisina eikä niitä ole suojattu käyttäjän tekemiltä virheiltä. Ammattilaistason ohjelmissa on kuitenkin aina otettava huomioon käyttäjän mahdolliset virheet ja hoidettava ne huolellisesti.

Loogiset operaattorit

Ohjelmissa joudutaan usein testaamaan useampia ehtoja samassa lauseessa. "Onko totta, että x on suurempi kuin y ja onko y samalla suurempi kuin z?" Koodin on ehkä testattava, ovatko molemmat ehdot tosia, jolloin suoritetaan joitakin toimintoja.

Kuvittele kehittynyttä hälytyssysteemiä: "Jos ovikello soi JA kello on yli 6.00 JA Ei ole pyhäpäivä TAI on viikonloppu, niin tee hälytys". Tässä lauseessa käytettiin jo kolmea erilaista loogista operaattoria. C++ -kielen loogiset operaattorit on esitetty taulukossa 4.2.

Taulukko 4.2. Loogiset operaattorit.

Operaattori	Symboli	Esimerkki
AND	&&	ilmaus1 && ilmaus2
OR	| |	ilmaus1 | | ilmaus2
NOT	!	!ilmaus

Looginen JA

Looginen JA-operaattori tutkii kahta lauseketta ja jos nuo lausekkeet ovat tosia, myös koko JA-lause on tosi. Jos on totta, että olet nälkäinen ja jos on totta, että sinulla on rahaa, voit mennä syömään lounasta. Täten

```
if ( (x == 5) && (y == 5) )
```

olisi tosi, jos sekä x että y ovat yhtäsuuria kuin 5 ja epätosi, jos jompikumpi muuttujista on eri suuri kuin 5. Huomaa, että molempien vertailuehtojen tulee olla tosia, jotta koko lauseke olisi tosi.

Looginen JA merkitään symbolein &&.

Looginen TAI

Looginen TAI koskee kahta vertailulauseketta. Jos jompikumpi on tosi, on TAI-lause myöskin tosi. Jos sinulla on rahaa TAI luottokortti, voit maksaa laskun. Sinulla ei tarvitse olla rahan lisäksi luottokorttia; tarvitset vain toisen, mutta molemmat olisi tietenkin hyvä olla olemassa.
Täten

```
if ( (x == 5) | | (y == 5) )
```

antaa arvon tosi, jos joko x tai y on yhtäsuuri kuin 5. Itse asiassa, jos x on 5, ei muuttujaa y testata lainkaan!

Looginen TAI merkitään symbolein | |.

Looginen EI

Looginen EI saa arvon tosi, jos testattava ehto on epätosi. Jos testattava ehto on epätosi, saa lause arvokseen tosi. Täten

```
if ( !(x == 5) )
```

on tosi vain, jos x on eri suuri kuin 5. Sama voitaisiin kirjoittaa näin

```
if ( x != 5)
```

Vertailujen suoritusjärjestys

Vertailuoperaattorit ja loogiset operaattorit palauttavat joko arvon 1 (tosi) tai 0 (epätosi). Myös niillä on suoritusjärjestyksensä, jonka mukaisessa järjestyksessä ne suoritetaan. Tämä on tärkeää seuraavanlaisen lauseen arvoa määritettäessä:

```
if ( x > 5 && y > 5 | | z > 5)
```

Mahdollisesti ohjelmoija halusi lauseen olevan tosi, jos sekä x että y ovat suurempia kuin 5 tai jos z on suurempi kuin 5. Toisaalta ohjelmoija saattoi haluta lauseen saavan arvon tosi vain, jos x on suurempi kuin 5 ja jos joko y tai z on suurempi kuin 5.

Jos x on 3 ja y ja z ovat 10, olisi ensimmäinen tulkinta tosi (z on suurempi kuin 5, joten ohitetaan x ja y), mutta toinen tulkinta taas olisi epätosi (x ei ole suurempi kuin 5, joten ei ole väliä, ovatko y ja z suurempia kuin 5).

Vaikkakin prioriteetit määräävät suoritusjärjestyksen, voivat sulkumerkit muuttaa järjestystä ja parantaa samalla luettavuutta:

```
if ( (x > 5) && (y > 5 | | z > 5) )
```

Edellä annettuja arvoja käyttäen olisi tämä lause epätosi. Koska x ei ole suurempi kuin 5, ei JA-operaattorin vasen puoli toteudu, joten koko lause on epätosi. Muistathan, että JA-operaattorin molempien osien tulee olla tosia, jotta koko JA-lauseke olisi tosi.

Huom! Usein on hyvä käyttää sulkumerkkejä koodin selkeyttämiseen ryhmittelyjen kautta. Muista, että tavoitteena on kirjoittaa toimivia

ohjelmia, joita on samalla helppo lukea ja ymmärtää.

Lisää totuusarvoista

C++ -kielessä on nolla epätosi ja nollasta poikkeava arvo tosi. Koska ilmauksilla on aina arvo, monet C++ -ohjelmoijat hyödyntävät tätä piirrettä if-lauseissaan. Seuraavanlainen lause

```
if (x)  // jos x on tosi (nollasta poikkeava)
    x = 0;
```

voidaan lukea: "jos x:llä on nollasta poikkeava arvo, aseta se nollaksi". Sama olisi voitu kirjoittaa hieman selvemmin seuraavasti:

```
if (x != 0) // jos x on nollasta poikkeava
    x = 0;
```

Molemmat lauseet ovat laillisia, mutta jälkimmäinen on selkeämpi. Hyvänä ohjelmointitapana on käyttää edellistä menettelyä todellisissa loogisissa testeissä eikä nollasta poikkeavien arvojen testaamisessa.

Seuraavat kaksi lausetta ovat myöskin yhdenmukaisia:

```
if (!x) // jos x on epätosi (nolla)
if (x == 0) // jos x on nolla
```

Jälkimmäinen lause on kuitenkin hieman ymmärrettävämpi ja selittävämpi.

Yhteenveto

Tässä luvussa opit tuntemaan ohjelmalauseen. Sait myös tietää, että ilmaus on mikä tahansa lause, joka palauttaa arvon. Opit myös, että välejä (välilyöntejä, sarkaimia, jne) voidaan käyttää ohjelmissa selkeyttämään koodia. Käsittelimme luvussa myös matemaattiset operaattorit ja sijoitusoperaattorin.

Vertailuoperaattoreiden lisäksi opit käyttämään etuliite- ja jälkiliiteoperaattoreita. Luvun lopussa käsittelimme if-lausetta, else-osaa sekä luotiin monimutkaisia ja sisäkkäisiä if-rakenteita.

Kysymyksiä ja vastauksia

K
Miksi sulkumerkkejä kannattaisi käyttää, koska prioriteetit määräävät operaattoreiden suoritusjärjestyksen?
V
Vaikka kääntäjä tietääkin suoritusjärjestyksen ja järjestys voidaan nähdä luettelosta, tekevät sulkumerkit koodista helpommin ymmärrettävää ja ylläpidettävää.

K
Jos vertailuoperaattorit palauttavat aina ykkösen tai nollan, miksi muita arvoja pidetään tosi-arvoina?
V
Vertailuoperaattorit palauttavat 1 tai 0, mutta jokainen ilmaus palauttaa arvon ja noita arvoja voidaan myöskin käyttää if-lauseessa. Seuraavassa on esimerkki:

```
if ( (x = a + b) == 35 )
```

Tuo lause on täysin laillinen. Se palauttaa arvon, vaikka muuttujien a ja b summa ei ole 35. Huomaa myös, että muuttujaan x sijoitetaan joka tapauksessa a:n ja b:n summa.

K
Mitä vaikutusta on välilyönneillä, sarkaimilla ja rivinvaihdoilla ohjelmaan?
V
Tulostumattomilla merkeillä ei ole vaikutusta ohjelmaan, mutta niiden avulla voidaan ohjelmakoodista tehdä luettavampaa.

K
Ovat negatiiviset luvut tosia vai epätosia?
V
Kaikki nollasta poikkeavat luvut ovat tosia.