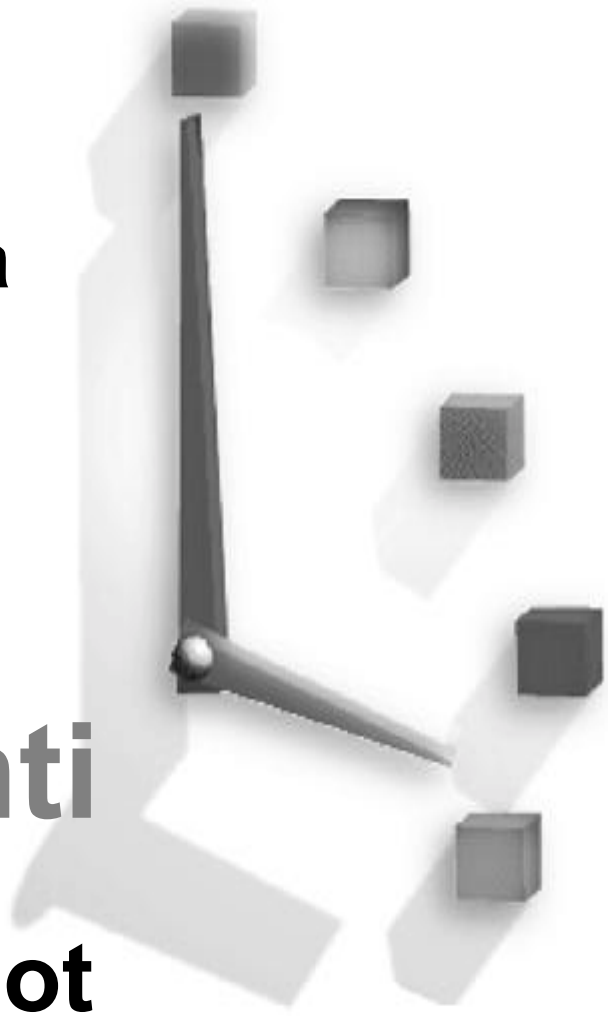
Osa

I



3. oppitunti

Muuttujat ja vakiot

Ohjelman tulee kyetä tallentamaan käyttämänsä tiedot. Muuttujat ja vakiot ovat keinoja työskennellä luvuilla ja muunlaisilla arvoilla.

Tässä luvussa käsitellään seuraavia asioita:

- ❏ Kuinka muuttujia ja vakioita esitellään ja määritellään
- ❏ Kuinka muuttujiin sijoitetaan arvoja ja kuinka noita arvoja käsitellään
- ❏ Kuinka muuttujan arvo tulostetaan näytölle

Mikä on muuttuja?

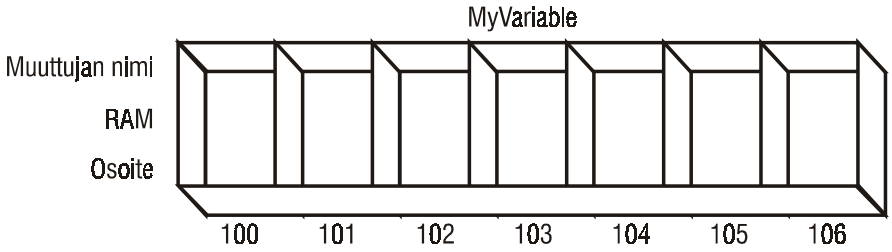
Uusi käsite Muuttuja on muistipaikka, johon voidaan tallentaa arvo, ja josta tuo arvo voidaan tarvittaessa hakea.

Jotta ymmärtäisit edellä kerrotun, on sinun tiedettävä, kuinka tietokoneen muisti toimii. Tietokoneen muistia voidaan pitää hyllynä, jossa on useita peräkkäisiä hyllypaikkoja. Kullakin hyllypaikalla on järjestysnumeronsa

samoin kuin muistipaikoillakin. Muistipaikkojen järjestysnumeroa kutsutaan muistiosoitteeksi.

Muuttujilla ei ole pelkästään osoitteita vaan niillä on myös nimet. Voit luoda esimerkiksi muuttujan nimeltä myAge. Tuo muuttujanimi tulee nyt otsikoksi johonkin hyllypaikkaan (vastaa muistipaikkaa) ja voit nyt hakea tavaraa tuosta hyllypaikasta (muistipaikasta) katsomatta paikan numeroa (muistiosoitetta).

Kuvassa 3.1 on kaaviokuva tilanteesta. Kuvassa on luotu muuttuja nimeltä myVariable ja se alkaa muistiosoitteesta 103.



Kuva 3.1. Kaaviokuva muistista.

Huom! Käyttömuistista (myös nimiä työmuisti, keskusmuisti, suorasaanti-muisti käytetään) puhuttaessa käytetään usein käsitettä RAM.

RAM tulee sanoista Random Access Memory. Kun ajat ohjelmaa, se ladataan RAM-muistiin levyltä. Myös kaikki muuttujat varaavat tilaa RAM-muistista. Kun ohjelmoijat puhuvat muistista, he tarkoittavat yleensä Ram-muistia.

Muistin varaaminen

Uusi käsite Kun määrittelet muuttujan C++ -kielessä, sinun on kerrottava kääntäjälle muuttujan nimi sekä se, millaisia arvoja muuttujaan tallennetaan: kokonaisluku, merkki, tms. eli muuttujan tyyppi. Muuttujan tyyppi kertoo kääntäjälle, kuinka paljon tilaa sen on varattava tallentamaan muuttujan arvo.

Kukin hyllypaikka on yhden tavun kokoinen. Jos luomasi muuttujan tyyppi vie tilaa kaksi tavua, on tilaakin varattava vastaava määrä eli nyt kaksi hyllypaikkaa. Muuttujan tyyppi (esimerkiksi int) kertoo kääntäjälle, kuinka paljon tilaa (kuinka monta hyllypaikkaa) sen on varattava muistista.

Koska tietokoneet käyttävät bittejä ja tavuja esittämään arvoja ja koska muistia mitataan tavuina, on tärkeää ymmärtää hyvin nuo käsitteet.

Kokonaislukujen koot

Kukin muuttuja vie tietokoneympäristöstä riippuvan vakiomäärän tilaa muistista. Esimerkiksi int-tyyppi vie jossakin tietokoneessa kaksi tavua ja toisessa ympäristössä neljä tavua.

Uusi käsite Merkkimuuttuja (char) on yleensä yhden tavun kokoinen. Tyyppi short int vie kaksi tavua useimmissa tietokoneissa, long int neljä tavua ja int kaksi tai neljä tavua. Jos käytät Windows 95- tai NT -käyttöjärjestelmiä (32-bittiset järjestelmät), on int-tyypin koko neljä tavua.

Listaus 3.1 auttaa löytämään noiden tietotyyppien tarkat koot juuri omassa koneessasi.

Listaus 3.1. Ohjelma kertoo tietotyyppien koot.

```
1:  #include <iostream.h>
2:
3:  int main()
4:  {
5:    cout << "The size of an int is:\t\t"    << sizeof(int)    << " bytes.\n";
6:    cout << "The size of a short int is:\t" << sizeof(short)  << " bytes.\n";
7:    cout << "The size of a long int is:\t"  << sizeof(long)   << " bytes.\n";
8:    cout << "The size of a char is:\t\t"    << sizeof(char)   << " bytes.\n";
9:    cout << "The size of a float is:\t\t"   << sizeof(float)  << " bytes.\n";
10:   cout << "The size of a double is:\t"    << sizeof(double) << " bytes.\n";
11:
12:   return 0;
13: }
```

Tulostus

```
The size of an int is:          2 bytes
The size of a short int is:     2 bytes
The size of a long int is:      4 bytes
The size of a char is:          1 bytes
The size of a float is:         4 bytes
The size of a double is:        8 bytes
```

Huomautus

Tavujen esittämistapa voi olla erilainen koneessasi!

Analyysi

Listaus 3.1 lienee melko selkeä. Ainoa uusi piirre on riveillä 5-10 oleva sizeof()-funktio, joka tulee kääntäjän mukana. Kyseinen funktio kertoo sille parametrina viedyn kohteen koon. Esimerkiksi rivillä 5 on sizeof()-funktion parametrina int-tyyppi. Tulostus kertoo, että omassa koneessani on int-tyypin koko sama kuin short int -tyypin eli kaksi tavua.

Etumerkillinen ja etumerkitön

Uusi käsite Lukuarvoja tallentavat tietotyypit voivat olla etumerkillisiä tai etumerkittömiä. Idea on siinä, että joskus tarvitset negatiivisia lukuja, joskus taas et. Kokonaisluvut (short ja long) ovat ilman unsigned-lisämäärettä etumerkillisiä. Lisämääreellä unsigned määritellyt kokonaisluvut ovat aina positiivisia.

Koska sekä etumerkilliset että etumerkittömät luvut vievät yhtä paljon tilaa, voi etumerkitön kokonaisluku olla kaksi kertaa suurempi kuin etumerkillinen. Tyyppiä unsigned short olevaan muuttujaan voidaan tallentaa lukuja väliltä 0 - 65535, kun taas tyyppiä signed short olevaan muuttujaan voidaan tallentaa lukuja väliltä -32768 - 32767.

Muuttujien perustyypit

C++ sisältää kokonaislukujen lisäksi monia muitakin tietotyyppejä. Tietotyypit jaetaan yleensä kokonaislukuihin, liukulukuihin ja merkkityyppeihin.

Liukulukumuuttujiin voidaan tallentaa desimaalilukuja. Merkkimuuttujat vievät tilaa yhden tavun ja niitä käytetään tallentamaan ASCII-taulukon 256 merkkiä tai symbolia.

Uusi käsite ASCII-merkistö on joukko standardimerkkejä tietokonekäyttöön. ASCII tulee sanoista American Standard Code for Information Interchange. Lähes kaikki tietokonejärjestelmät tukevat ASCII-koodia, vaikkakin monet tukevat myös muita kansainvälisiä merkistöjä.

C++ -kielen muuttujatyypit on kuvattu taulukossa 3.1. Taulukosta nähdään muuttujan tyyppi, sen tilantarve ja erilaiset arvot, joita kyseiseen muuttujatyyppiin voidaan tallentaa. Tallennettavat arvot riippuvat aina kyseisen tietotyypin viemästä muistitilasta, joka tulisi aina tarkistaa konekohtaisesti eli esimerkiksi ajamalla listaus 3.1 omassa koneessa.

Taulukko 3.1. Muuttujatyypit.

Tyyppi	Koko	Arvot
Unsigned short int	2	0-65535
Short int	2	-32768-32767
Unsigned long int	4	0-4294967295
Long int	4	-2147483648-2147483647
Char	1	256 eri merkkiä
Float	4	1.2e-38 - 3.4e38
Double	8	2.2e-308-1.8e308

Muuttujan määrittely

Muuttuja luodaan tai määritellään antamalla ensin sen tyyppi ja sen jälkeen välilyönti, jota seuraa muuttujan nimi ja puolipiste. Muuttujan nimi voi olla mikä tahansa kirjainyhdistelmä, mutta siinä ei saa olla välilyöntejä. Hyväksyttyjä muuttujan nimiä ovat esimerkiksi x, J23qrsnf ja myAge. Hyvä muuttujan nimi kertoo muuttujan tarkoituksen, mikä lisää luettavuutta koodiin. Seuraava ohjelmalause määrittää kokonaislukumuuttujan nimeltä myAge:

```
int myAge;
```

On hyvä ohjelmointitapa antaa muuttujille kuvaavat nimet. Sellaiset nimet kuin myAge ja howMany helpottavat koodin lukemista paremmin kuin nimet likexJ4 tai theInt. Kun käytät kuvaavia muuttujanimiä, voit vähentää kommenttien käyttöä.

Tee seuraava koe: arvaa, mitä seuraavat ohjelman osat tekevät muutaman rivin perusteella:

Esimerkki 1

```
main()
{
  unsigned short x;
  unsigned short y;
  unsigned short z;
  z = x * y;
}
```

Esimerkki 2

```
main()
{
  unsigned short Width;
  unsigned short Length;
  unsigned short Area;
  Area = Width * Length;
}
```

Isot ja pienet kirjaimet

Uusi käsite C++ tunnistaa isot ja pienet kirjaimet. Toisin sanoen isot ja pienet kirjaimet ovat eri merkkejä C++ -kielessä. Muuttuja nimeltä age on eri muuttuja kuin Age, joka taas on eri muuttuja kuin AGE.

Huom! Jotkut kääntäjät sallivat kääntää isojen ja pienten kirjainten erottelun pois päältä. Älä kuitenkaan tee niin, koska tällöin koodisi ei toimi muissa kääntäjissä eivätkä toiset ohjelmoijat enää ymmärrä koodiasi.

Monet ohjelmoijat käyttävät pikkukirjaimia muuttujien nimissä. Jos nimi vaatii kaksi sanaa (esimerkiksi my car), käytetään yleisesti kahta eri menettelytapaa muuttujan nimeämisessä: my_car tai myCar. Jälkimmäistä menettelyä kutsutaan kamelinotaatioksi, koska iso kirjain aiheuttaa ikään kuin kyttyrän nimeen.

Varatut sanat

C++ on varannut joitakin sanoja omaan käyttöönsä eikä niitä pidä käyttää muuttujien nimissä. Nuo varatut sanat kuuluvat ohjelmointikieleen. Tällaisia varattuja sanoja ovat muun muassa if, while, for ja main. Kääntäjämanuaalissasi on lueteltu kaikki kääntäjäsi varatut sanat. Pyri ensisijaisesti käyttämään kuvaavia muuttujanimiä.

Tee/Älä tee

Määritä muuttuja antamalla tyyppi ja sitten nimi.
Käytä merkityksellisiä nimiä.
Muista, että C++ tunnistaa isot ja pienet kirjaimet.
Älä käytä C++:n varattuja sanoja muuttujanimissä.
Ymmärrä, mitä muuttujatyypin varaama muistitila merkitsee ja pyri käyttämään muistia säästeliäästi.
Älä käytä etumerkittömiä tyyppejä negatiivisten lukujen kohdalla.

Useamman muuttujan luominen samalla kertaa

Yhdessä lauseessa voidaan luoda samalla useampia muuttujia. Tällöin kirjoitetaan ensin tyyppi ja sitten kyseistä tyyppiä olevat muuttujanimet pilkuilla erotettuina.

Esimerkki:

```
unsigned int myAge, myWeight;   // kaksi etumerkitöntä koko
                                // naislukumuuttujaa
long area, width, length;  // kolme long-muuttujaa
```

Kuten näet, on myAge ja myWeight määritelty etumerkittöminä kokonaislukumuuttujina. Toinen rivi määrittelee kolme yksittäistä long-muuttujaa nimeltä area, width ja length. Tyyppi long liitetään kaikkiin noihin kolmeen muuttujaan eikä tyyppejä voida sekoittaa yhdessä ohjelmalauseessa.

Arvojen sijoittaminen muuttujiin

Muuttujaan sijoitetaan arvo käyttämällä sijoitusoperaattoria (=). Täten muuttujaan Width sijoitetaan arvo 5 kirjoittamalla

```
unsigned short Width;
Width = 5;
```

Määrittely ja sijoittaminen voidaan yhdistää (alustaminen) kirjoittamalla:

```
unsigned short Width = 5;
```

Alustaminen näyttää aivan sijoittamiselta eikä kokonaislukumuuttujien kohdalla eroja juuri olekaan. Myöhemmin, kun tutustumme vakioihin, huomaat, että joskus alustus on pakko tehdä.

Listaus 3.2 on kokonainen ohjelma ja valmis käännettäväksi. Se laskee suorakaiteen alan ja tulostaa sen näytölle.

Listaus 3.2. Muuttujien esittely.

```
1:  // Muuttujien esittely
2:  #include <iostream.h>
3:
4:  int main()
5:  {
6:    int Width = 5, Length;
7:    Length = 10;
8:
9:    // Luodaan int-muuttuja ja alustetaan
10:   int Area  = Width * Length;
11:
12:   cout << "Width:" << Width << "\n";
13:   cout << "Length: "  << Length << endl;
14:   cout << "Area: " << Area << endl;
15:   return 0;
16: }
```

Tulostus

```
Width: 5
Length: 10
Area: 50
```

Analyysi

Rivillä 2 on include-komento, joka lisää koodiin kirjastotiedoston, joka hoitaa syöttö- ja tulostusvirrat. Rivi 4 aloittaa itse ohjelman.
Rivillä 6 määritellään int-tyyppinen muuttuja Width ja alustetaan se arvolla 5. Toinen int-muuttuja Length jätetään alustamatta. Rivillä 7 sijoitetaan muuttujaan Length arvo 10. Rivillä 10 määritellään int-muuttuja Area ja alustetaan se muuttujien Width ja Length tulolla. Riveillä 12-15 tulostetaan muuttujien arvot näytölle. Huomaa erikoissana endl, joka luo uuden rivin.

typedef

Uusi käsite Joskus voi tuntua turhalta ja aikaavievältä ja kaiken lisäksi virhealttiilta kirjoittaa määrityksiä unsigned short int useaan kertaan. C++ mahdollistaakin sijaisnimien luomisen sanaryhmille. Tällöin käytetään varattua sanaa typedef (joka on lyhennelmä sanoista type definition) ja annetaan muuttujalle toinen nimi.

Itse asiassa typedef-sanalla luodaan synonyymi eikä siis uutta muuttujatyyppiä. Uusia muuttujia luodaan luvussa 6, "Perusluokat". Sijaisnimi luodaan kirjoittamalla ensin varattu sana typedef ja sen jälkeen olemassa oleva tyyppi ja lopuksi uusi nimi. Seuraavassa on esimerkki:

typedef unsigned short int USHORT

Lause antaa muuttujatyypille unsigned short int uuden nimen USHORT, jota voidaan tämän jälkeen käyttää määrittelyissä. Listaus 3.3 on listauksen 3.2 versio, jossa käytetään tietotyyppien sijaisnimiä.

Listaus 3.3. Esittelee typedef-käyttöä.

```
1:  // *****************
2:  // Esitellään typedef-määre
3:  #include <iostream.h>
4:
5:  typedef unsigned short int USHORT;       //typedef määritelty
6:  typedef unsigned long ULONG;
7:  int main()
8:  {
9:    USHORT  Width = 5;
10:   USHORT Length;
11:   Length = 10;
12:   ULONG Area  = Width * Length;
13:   cout << "Width:" << Width << "\n";
14:   cout << "Length: "  << Length << endl;
15:   cout << "Area: " << Area <<endl;
16:   return 0;
17: }
```

Tulostus

```
Width: 5
Length: 10
Area: 50
```

Analyysi

Huom! Joissakin kääntäjissä tämä koodi aiheuttaa varoituksen, että muunnoksessa saatetaan menettää merkitseviä numeroita. Se johtuu siitä, että rivillä 12 oleva kahden USHORT-muuttujan tulo saattaa

olla suurempi kuin etumerkitön short-muuttuja voi tallentaa. Tällöin tulosta saatetaan joutua katkaisemaan numeroita. Esimerkkiohjelmassamme ei kuitenkaan ole tällaista vaaratekijää.

Rivillä 5 annetaan tyypille unsigned short int synonyymi USHORT. Muutoin ohjelma on samanlainen kuin listaus 3.2.

Milloin tulisi käyttää short- ja milloin long-tyyppiä

Eräs aloittelevan ohjelmoijan ongelma on tietää, milloin muuttujan tyyppinä tulisi käyttää short- ja milloin taas long-tyyppiä. Sääntö on yksinkertainen: jos muuttujaan tallennettava arvo on jossakin tilanteessa suurempi kuin muuttujan tyyppi sallii, käytä suurempaa tyyppiä.

Kuten taulukosta 3.1 nähdään, voidaan kaksitavuisiin unsigned short int -muuttujiin tallentaa maksimiarvo 65535 ja etumerkilliseen (signed) muuttujaan puolta pienempi arvo. Vaikkakin unsigned long int -muuttujiin voidaan tallentaa erittäin suuria lukuja, on se kuitenkin suurin mahdollinen koko kokonaisluvuille. Jos luvut ovat erittäin suuria, on siirryttävä float- ja double-muuttujiin. Tällöin saatetaan kuitenkin menettää tarkkuutta. Float- ja double-muuttujiin voidaan tallentaa suuria lukuja, mutta vain ensimmäiset 7 tai 19 numeroa ovat merkitseviä useimmissa tietokoneissa. Jos numeroita on enemmän, suoritetaan pyöristäminen.

Etumerkittömien kokonaislukujen arvoalueen pyörähtäminen ympäri

Etumerkittömien kokonaislukujen arvoalueen rajallisuus on harvoin ongelma, mutta mitä tapahtuu, kun ylität arvoalueen?

Kun maksimiarvo ylitetään, aloitetaan uusi kierros aivan kuin auton matkamittarissa. Listaus 3.4 osoittaa, mitä tapahtuu yritettäessä sijoittaa liian suurta arvoa short int -tyyppiseen muuttujaan.

Listaus 3.4. Liian suuren arvon sijoittaminen unsigned-muuttujaan.

```
1:  #include <iostream.h>
2:  int main()
3:  {
4:    unsigned short int smallNumber;
5:    smallNumber = 65535;
6:    cout << "small number:" << smallNumber << endl;
7:    smallNumber++;
8:    cout << "small number:" << smallNumber << endl;
9:    smallNumber++;
10:   cout << "small number:" << smallNumber << endl;
11:   return 0;
```

```
12: }
```

Tulostus

```
small number: 65535
small number: 0
small number: 1
```

Analyysi

Rivillä 4 esitellään unsigned short int -tyyppinen muuttuja, smallNumber, joka omassa koneessani on kaksitavuinen ja voi tallentaa arvoja väliltä 0 - 65535. Rivillä 5 sijoitetaan muuttujaan maksimiarvo, joka tulostetaan rivillä 6.

Rivillä 7 kasvatetaan muuttujan arvoa yhdellä. Kasvatuksen symboli on ++ -merkintä (kuten sanassa C++ - C-kieltä on edelleen kasvatettu). Niinpä muuttujan arvona tulisi nyt olla 65536. Mutta unsigned short int -tyyppinen kokonaisluku ei voi tallentaa niin suurta arvoa, joten kierros aloitetaan alusta ja muuttuja saa arvon 0, joka tulostetaan rivillä 8.

Rivillä 9 kasvatetaan muuttujan arvoa uudelleen ja seuraavalla rivillä tulostetaan uusi arvo, 1.

Etumerkillisen kokonaisluvun arvoalueen pyörähtäminen ympäri

Etumerkillinen kokonaisluku on hieman erilainen kuin etumerkitön nimenomaan negatiivisten arvojen kohdalla. Se ei toimi matkamittarin tavoin vaan siirtyy positiivisten arvojen esittämisen jälkeen esittämään negatiivisia arvoja. Yksi pykälä nollasta eteenpäin on joko 1 tai -1. Kun positiiviset numerot ovat loppuneet, aletaan näyttää suurimpia negatiivisia numeroita ja kuljetaan sitten kohti nollaa. Listaus 3.5 osoittaa, mitä tapahtuu, kun lisäät ykkösen signed short int -tyyppisen muuttujan maksimiarvoon.

Listaus 3.5. Liian suuren arvon sijoittaminen signed int -tyyppiseen muuttujaan.

```
1:  #include <iostream.h>
2:  int main()
3:  {
4:    short int smallNumber;
5:    smallNumber = 32767;
6:    cout << "small number:" << smallNumber << endl;
7:    smallNumber++;
8:    cout << "small number:" << smallNumber << endl;
9:    smallNumber++;
10:   cout << "small number:" << smallNumber << endl;
11:   return 0;
12: }
```

Tulostus

```
small number: 32767
small number: -32768
small number: -32767
```

Analyysi

Rivillä 4 esitellään etumerkillinen kokonaislukumuuttuja, smallNumber. Jos muuttujaa ei määritellä etumerkittömäksi, se on oletuksena etumerkillinen. Ohjelma jatkaa kuten edellisessä listauksessa, tulostus on kuitenkin erilainen. Jotta ymmärtäisit, mitä tapahtuu, sinun tulee nähdä arvojen esitystapa bitteinä. Katso lisätietoa liitteestä C, "Binääri- ja heksadesimaaliluvut".

Viimeinen tulostusrivi toimii samoin kuin etumerkittömien kokonaislukujen kohdalla: etumerkillinen kokonaisluku pyörähtää ympäri suurimmasta positiivisesta maksimiarvosta suurimpaan negatiiviseen arvoon.

Vakiot

Uusi käsite Muuttujien lailla myös vakiot ovat tiedon tallennuspaikkoja. Mutta muuttujien arvot voivat vaihdella ohjelman ajon aikana, kun taas vakioiden arvot pysyvät samoina.

Vakio täytyy alustaa esittelyn yhteydessä eikä siihen voida myöhemmin sijoittaa uutta arvoa.

Literaalit vakiot

Uusi käsite C++ -kielessä on kahdentyyppisiä vakioita: literaalit ja symboliset.

Literaali vakio on kirjoitettu arvo. Esimerkiksi lauseessa

```
int myAge = 39;
```

on myAge muuttuja ja 39 literaali vakio.

Symboliset vakiot

Symbolinen vakio on vakio, jota esittää nimi kuten muuttujaakin. Vakiota ei kuitenkaan voida muuttaa alustamisen jälkeen.

Jos ohjelmassasi on kokonaislukumuuttujat nimeltä students ja classes, voit laskea oppilaiden lukumäärän, jos tiedät, että kullakin luokalla (classes) on 15 oppilasta:

```
students = classes * 15;
```

* merkitsee kertolaskua.

Huom!

Esimerkkilauseessa on 15 literaali vakio. Koodisi olisi helppolukuisempaa ja paremmin ylläpidettävää, jos korvaisit arvon symbolisella vakiolla:

```
students = classes * studentsPerClass;
```

Jos päättäisit myöhemmin muuttaa kullakin luokalla olevien oppilaiden määrää, voisit tehdä sen vakion alustamisen yhteydessä eikä muutosta tarvitsisi tehdä eri puolilla ohjelmakoodia.

Vakioiden määrittely #define-komennolla

Vanhanaikainen ja huono tapa määrittää vakio on käyttää #define-komentoa:

```
#define studentsPerClass 15
```

Huomaa, että vakio ei ole mitään erityistä tietotyyppiä (int, char, tms.). #define suorittaa yksinkertaisen tekstikorvauksen. Aina, kun esikäsittelijä näkee sanan studentsPerClass, se sijoittaa arvon 15 tekstiin.

Koska esikäsittelijä ajetaan ennen kääntämistä, ei kääntäjä näe koskaan vakiota, vaan ainoastaan arvon 15.

Vakioiden määrittely const-avainsanalla

Vaikkakin #define toimii oikein, on vakioiden määrittelylle uusi, parempi, nopeampi ja hienostuneempi menettelytapa:

```
const unsigned short int studentsPerClass = 15;
```

Esimerkkimme määrittelee myöskin symbolisen vakion nimeltä studentsPerClass, mutta nyt vakio on tyyppiä unsigned short int.

Kirjoitustapa on pitempi, mutta se tarjoaa useita etuja. Suurin ero on siinä, että vakiolla on tyyppi ja kääntäjä voi tarkistaa, että vakiota käytetään tyyppinsä mukaisesti.

Luetellut (enumerated) tyypit

Luetellut vakiot ovat joukko vakioita, joilla on tietyt arvot. Voit määrittää esimerkiksi vakion COLOR luetelluksi vakioksi ja antaa sen saada arvot RED, BLUE, GREEN, WHITE ja BLACK.

Lueteltujen vakioiden syntaksissa kirjoitetaan ensin varattu sana enum, jota seuraavat tyypin nimi, aloittava aaltosulku, jäsenet pilkuilla erotettuina ja lopuksi luettelon päättävä aaltosulku. Seuraavassa on esimerkki:

```
Enum COLOR = { RED, BLUE, GREEN, WHITE, BLACK};
```

Lause suorittaa kaksi tehtävää:
1. Sanasta COLOR tulee luettelon nimi eli uusi tyyppi
2. RED on symbolinen vakio, joka saa arvon 0, BLUE vastaavasti arvon 1, jne.

Jokaisella luetellulla vakiolla on kokonaislukuarvo. Jos arvoa ei määritellä erikseen, on ensimmäisen vakion arvona 0 ja muut lasketaan siitä ylöspäin. Mikä tahansa vakio voidaan kuitenkin alustaa tietyllä arvolla ja alustamisen jälkeisten vakioiden arvot lasketaan alustetusta arvosta ylöspäin. Niinpä, jos esittely on seuraavanlainen:

```
Enum COLOR = { RED=100, BLUE, GREEN=500, WHITE, BLACK=700};
```

vakion RED arvo on 100, vakion BLUE arvo 101, vakion GREEN arvo 500, vakion WHITE arvo 501 ja vakion BLACK arvo 700.

Yhteenveto

Tässä luvussa tutustuimme numeerisiin ja merkkimuuttujiin sekä vakioihin.

Muuttuja on esiteltävä ennen niiden käyttöä ja muuttujiin on sijoitettava vain määritellyn tietotyypin mukaisia arvoja. Jos kokonaislukumuuttujaan sijoitetaan liian suuria arvoja, saadaan virheellisiä tuloksia.

Luvussa käsiteltiin myös literaaleja ja symbolisia vakioita sekä lueteltuja tyyppejä. Symbolisia vakioita voidaan määritellä kahdella tavalla: #define-komennolla ja const-avainsanalla.

Kysymyksiä ja vastauksia

K
Jos short int ei riitä, miksei jatkuvasti käytetä long-tyyppiä?
V
Sekä short- että long-tyyppiset kokonaisluvut voivat olla riittämättömiä, mutta long-tyyppiin mahtuu huomattavasti suurempia lukuja. Kuitenkin long-tyyppi vie useimmissa järjestelmissä kaksi kertaa enemmän muistitilaa kuin short-tyyppi. Viime vuosina muistin niukkuus ei kuitenkaan ole enää ollut ongelmana.

K
Mitä tapahtuu, jos sijoitan desimaaliluvun kokonaislukumuuttujaan? Katso seuraavaa koodia:

```
int aNumber = 5.4;
```

V
Hyvä kääntäjä antaa virheilmoituksen, mutta sijoitus on täysin laillinen. Desimaaliluku katkaistaan kokonaisluvuksi eli muuttuja saa nyt arvon 5. Osa informaatiosta siis katoaa, vaikka sijoittaisitkin arvon tämän jälkeen float-muuttujaan.

K
Eikö voitaisi käyttää pelkästään literaaleja vakioita; miksi nähdä vaivaa symbolisten vakioiden kanssa?

V
Jos käytät arvoa monessa kohtaa ohjelmaa, voidaan symbolisella vakiolla tehdä kaikkien kohtien muutos pelkästään muuttamalla vakion alustusarvoa. Symbolisilla vakioilla on myös kuvaavat nimensä, mikä lisää koodin luettavuutta. Ajattele esimerkiksi vakiota nimeltä paiviaVuodessa lukuarvon 365 sijaan.