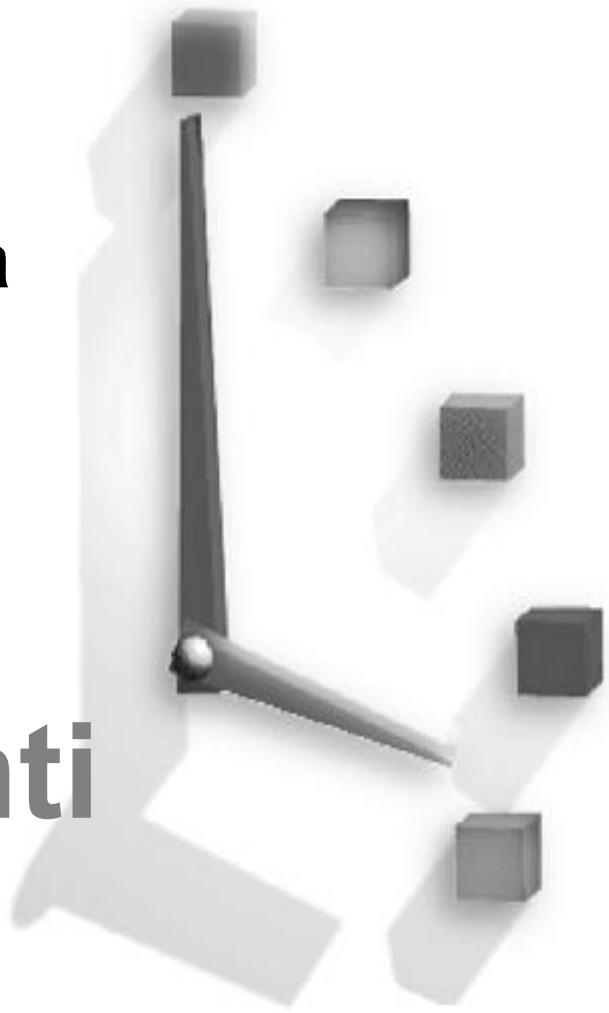
Osa

I



5. oppitunti

Funktiot

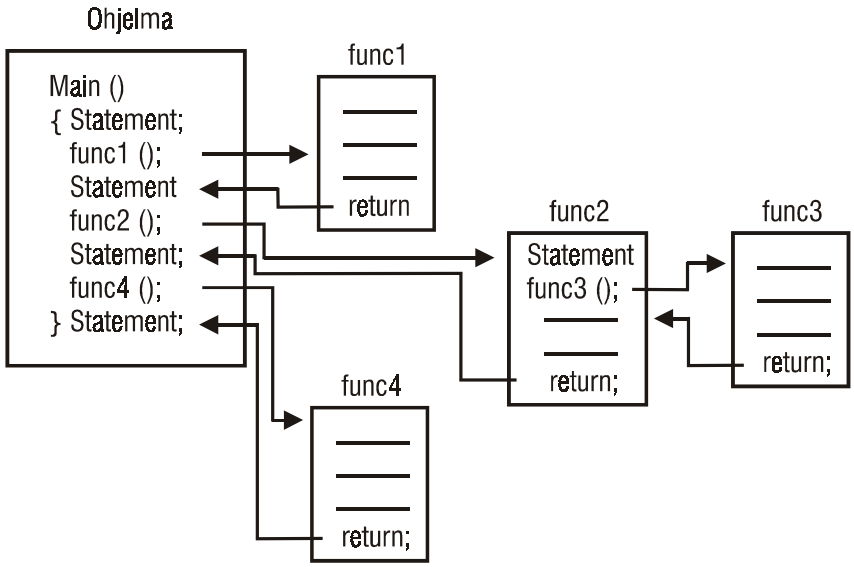
Kun C++ -kielestä puhutaan, mainitaan yleensä oliot ensimmäiseksi. Oliot taasen ovat riippuvia funktioista, joiden kautta ne toimivat. Tässä luvussa opit seuraavat asiat:

- ❏ Mikä on funktio ja mitkä ovat sen osat
- ❏ Kuinka funktio esitellään ja määritellään
- ❏ Kuinka parametreja viedään funktioon
- ❏ Kuinka funktiosta palautetaan arvo

Mikä on funktio?

Funktio on aliohjelma, joka voi käsitellä tietoa ja palauttaa arvon. Jokaisessa C++ -ohjelmassa on ainakin yksi funktio, main(). Kun ohjelma käynnistyy, kutsutaan main()-funktiota automaattisesti. Main() saattaa kutsua muita funktioita, jotka taas voivat edelleen kutsua muita funktioita.

Jokaisella funktiolla on nimensä ja kun tuo nimi kohdataan ohjelmassa, haaraudutaan suorittamaan funktion koodia. Kun funktio päättyy, suoritus palaa kutsun jälkeiselle riville. Tätä ohjelman kulkua on havainnollistettu kuvassa 5.1.



Kuva 5.1. Kun ohjelma kutsuu funktiota, suoritus siirtyy funktion runkoon, jonka jälkeen palataan kutsua seuraavalle riville.

Hyvin kehitetty funktio suorittaa tietyn tehtävän. Se tekee siis yhden tehtävän, mahdollisimman kokonaisesti ja päättyy sen jälkeen.

Monimutkaiset toiminnot tulisi jakaa useisiin funktioihin, jotka sitten kutsuvat toisiaan. Tämä tekee ohjelmasta helpomman ymmärtää ja ylläpitää.

Kirjassa puhutaan usein ylläpidettävyyden helpottamisesta. Ohjelman hinta ei koostu pelkästään koodaamisesta vaan koko sen eliniän jatkuvasta ylläpidosta ja luotettavuuden takaamisesta.

Funktioiden esittely ja määrittely

Ennen funktion käyttömahdollisuutta on funktio esiteltävä ja sitten määriteltävä.

Uusi käsite Esittelyssä kerrotaan kääntäjälle funktion nimi, palautustyyppi sekä parametrit. Funktion esittelyä kutsutaan funktion prototyypiksi.

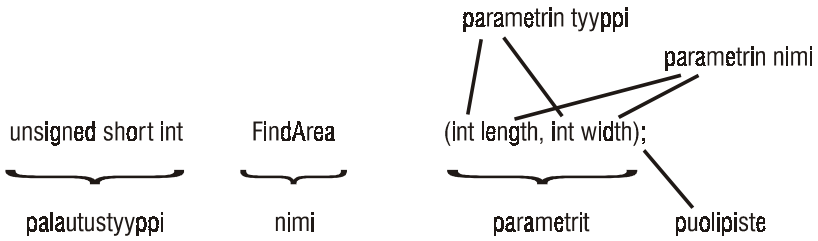
Uusi käsite Määrittely kuvaa kääntäjälle funktion toiminnan. Mikään funktio ei voi kutsua funktiota, jota ei ole määritelty.

Funktion esittely

Kääntäjän mukana tulevien funktioiden prototyypit ovat jo valmiina. Nuo funktiot saadaan käyttöön #include-komennolla.

Uusi käsite Funktion prototyyppi on lause, joka päättyy puolipisteeseen. Se sisältää funktion palautustyypin, nimen ja parametriluettelon.

Parametriluettelo on luettelo kaikista parametreista ja niiden tyypeistä pilkuilla erotettuina. Kuva 5.2 havainnollistaa funktion prototyypin osia.



Kuva 5.2. Funktion prototyypin osat.

Funktion prototyypin ja määrittelyn täytyy täsmätä täysin palautustyypin, nimen ja parametrien suhteen. Jos näin ei ole, generoi kääntäjä virheilmoituksen. Huomaa kuitenkin, että funktion prototyypin ei tarvitse sisältää parametrien nimiä, vaan parametrien tyypit riittävät. Seuraava prototyyppi on täysin laillinen:

```
long Area(int, int);
```

Tämä prototyyppi esittelee funktion nimeltä Area, joka palauttaa long-tyypin ja jolla on kaksi kokonaislukuparametria. Vaikkakin esittely on hyväksyttävä, se ei ole kovinkaan hyvä. Parametrien nimien lisääminen selkeyttää koodia. Sama funktio voitaisiin esitellä siis näinkin:

```
long Area(int length, int width);
```

Nyt funktion toiminta ja parametrit ovat selvästi tiedossa.

Huomaa, että kaikilla funktioilla on palautustyyppi. Listaus 5.1 esittelee ohjelman, joka sisältää Area()-funktion prototyypin.

Listaus 5.1. Funktion esittely ja määrittely sekä käyttö.

```
1:  // Listaus 5.1 - esittelee funktion prototyypin
2:  typedef unsigned short USHORT;
3:  typedef unsigned long ULONG;
4:  #include <iostream.h>
5:  ULONG FindArea(USHORT length, USHORT width); //funktion prototyyppi
6:
7:  int main()
8:  {
9:    USHORT lengthOfYard;
```

```
10:    USHORT widthOfYard;
11:    ULONG areaOfYard;
12:
13:    cout << "\nHow wide is your yard? ";
14:    cin >> widthOfYard;
15:    cout << "\nHow long is your yard? ";
16:    cin >> lengthOfYard;
17:
18:    areaOfYard= FindArea(lengthOfYard,widthOfYard);
19:
20:    cout << "\nYour yard is ";
21:    cout << areaOfYard;
22:    cout << " square feet\n\n";
23:    return 0;
24: }
25:
26: ULONG FindArea(USHORT l, USHORT w)
27: {
28:   return l * w;
29: }
```

Tulostus

```
How wide is your yard?  100
How long is your yard?  200
Your yard is 20000 square feet
```

Analyysi

FindArea()-funktion prototyyppi on rivillä 5. Vertaa prototyyppiä rivin 26 määrittelyyn. Huomaa, että nimi, palautustyyppi ja parametrit ovat samat.

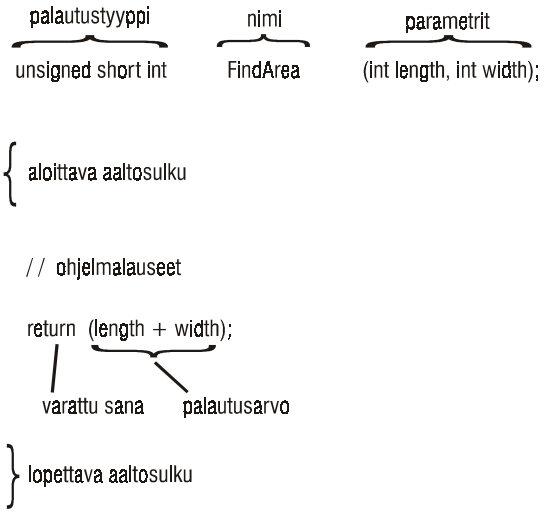
Jos ne olisivat erilaiset, olisi generoitu kääntäjän virhe. Itse asiassa ainoat erot ovat siinä, että funktion prototyyppi päättyy puolipisteeseen eikä siinä ole funktion runkoa.

Huomaa myös, että prototyypin parametrien nimet ovat length ja width, mutta määrittelyssä käytetään nimiä l ja w. Prototyypin parametrinimiä ei siis käytetä, vaan ne ovat pelkästään informaation antajia ohjelmoijalle. Argumentit viedään funktiolle siinä järjestyksessä kuin ne on esitelty ja määritelty.

Funktion määrittely

Funktion määrittely koostuu otsikosta ja rungosta. Otsikko on täsmälleen sama kuin prototyyppi lukuunottamatta parametrien nimiä, jotka tulee otsikossa mainita, sekä prototyypin tarvitsemaa puolipistettä.

Funktion runko on joukko ohjelmalauseita aaltosulkujen sisällä. Kuva 5.3 esittää funktion otsikon ja rungon.



Kuva 5.3. Funktion otsikko ja runko.

Funktiot

Funktion prototyyppi kertoo kääntäjälle funktion nimen, palautusarvon ja parametrit.

```
Palautustyyppi funktionnimi ( [tyyppi [parametrinimi]]…);
```

Funktion määrittely kuvaa funktion toiminnan.

```
Palautustyyppi funktionnimi ( [tyyppi parametrinimi]…)
{
    ohjelmalauseet;
}
```

Funktion prototyyppi kertoo kääntäjälle funktion palautustyypin, nimen ja parametriluettelon. Funktioiden ei ole pakko ottaa parametreja. Jos parametreja kuitenkin tarvitaan, ei prototyypissä ole pakko kertoa parametrien nimiä vaan tyypit riittävät. Prototyyppi päättyy aina puolipisteeseen (;).

Funktion määrittelyn otsikon tulee olla yhdenmukainen prototyypin kanssa. Siinä tulee olla kaikkien parametrien nimet. Runko sijoitetaan aaltosulkuihin. Kaikki rungon lauseet päättyvät puolipisteeseen, mutta itse funktio ei pääty puolipisteeseen vaan sulkevaan aaltosulkuun.

Jos funktio palauttaa arvon, sen tulisi päättyä return-lauseeseen, vaikkakin return-lauseet voivat sijaita millä kohtaa funktion runkoa tahansa.

Jokaisella funktiolla on palautustyyppi. Jos sitä ei ole erikseen määrätty, on palautustyyppinä int. Varmista, että jokaisella funktiollasi on

palautustyyppinsä. Jos funktio ei palauta arvoa, sen palautustyyppinä on void.

Seuraavassa on esimerkkejä funktion prototyypeistä:

```
long FindArea(long length, long width); //palauttaa long-
tyypin, 2 parametria
void printMessage(int messageNumber);  // palauttaa voidin, 1
parametri
int GetChoice();  // palauttaa int-arvon, ei parametreja
BadFunction();  // palauttaa int-arvon, ei parametreja
```

Seuraavassa on esimerkkejä funktioiden määrittelyistä:

```
long Area(long l, long w)
{
  return l * w;
}

void PrintMessage(int whichMsg)
{
  if (whichMsg == 0)
     cout << "Hello.\n";
if (whichMsg == 1)
     cout << "Goodbye.\n";
if (whichMsg > 1)
     cout << "I'm confused\n";
}
```

Paikalliset muuttujat

Uusi käsite Funktioon voidaan muuttujien viemisen lisäksi määritellä myös omia muuttujia. Omat muuttujat määritellään siis funktion rungossa ja niitä kutsutaankin paikallisiksi muuttujiksi. Kun funktio päättyy, eivät paikalliset muuttujat ole enää käytettävissä.

Paikalliset muuttujat määritellään aivan samoin kuin muutkin muuttujat. Myös funktioille vietyjä parametreja pidetään paikallisina muuttujina, jolloin niitä voidaan käyttää aivan kuin ne olisi määritelty funktion rungossa. Listaus 5.2 on esimerkki parametrien ja paikallisesti määriteltyjen muuttujien käyttämisestä.

Listaus 5.2. Paikallisten muuttujien ja parametrien käyttäminen.

```
1:  #include <iostream.h>
2:
3:  float Convert(float);
4:  int main()
5:  {
```

```
6:    float TempFer;
7:    float TempCel;
8:
9:    cout << "Please enter the temperature in Fahrenheit: ";
10:   cin >> TempFer;
11:   TempCel = Convert(TempFer);
12:   cout << "\nHere's the temperature in Celsius: ";
13:   cout << TempCel << endl;
14:   return 0;
15: }
16:
17: float Convert(float TempFer)
18: {
19:   float TempCel;
20:   TempCel = ((TempFer - 32) * 5) / 9;
21:   return TempCel;
22: }
```

Tulostus

```
Please enter the temperature in Fahrenheit: 212
Here's the temperature in Celsius: 100

Please enter the temperature in Fahrenheit: 32
Here's the temperature in Celsius: 0

Please enter the temperature in Fahrenheit: 85
Here's the temperature in Celsius: 29.4444
```

Analyysi

Riveillä 6 ja 7 esitellään kaksi float-muuttujaa, toinen tallentamaan Fahrenheit-lukema ja toinen taas Celsius-lukema. Käyttäjää kehoitetaan antamaan Fahrenheit-lukema rivillä 9 ja tuo arvo viedään funktiolle Convert().

Suoritus hyppää funktion Convert() ensimmäiselle riville (rivi 18), jossa esitellään myös paikallinen muuttuja nimeltä TempCel. Huomaa, että tämä muuttuja ei ole sama kuin rivin 7 TempCel, vaan se on käytettävissä vain Convert()-funktion sisällä. Parametrina TempFer viety arvo on myös vain paikallinen kopio muuttujasta.

Funktio olisi voinut nimetä parametrin FerTemp ja paikallisen muuttujan CelTemp ja ohjelma olisi toiminut yhtä hyvin. Voit antaa nuo nimet uudelleen ja testata ohjelman toiminnan.

Paikalliseen muuttujaan TempCel sijoitetaan laskettu Celsius-lukema ja se palautetaan sitten funktion palautusarvona ja sijoitetaan rivillä 11 main()-funktion muuttujaan TempCel. Rivillä 12 tulostetaan muuttujan arvo.

Ohjelma ajetaan kolme kertaa. Ensiksi viedään muunnosfunktiolle arvo 212, jotta nähdään funktion antavan veden kiehumispisteen oikein. Toiseksi testataan veden jäätymispiste ja kolmanneksi annetaan satunnainen lukema, joka tuottaa murto-osia.

Kokeile harjoituksena kirjoittaa ohjelmaan muita muuttujanimiä kuten seuraavassa:

```
#include <iostream.h>
float Convert(float);
int main()
{
float TempFer;
float TempCel;
cout << "Please enter the temperature in Fahrenheit: ";
cin >> TempFer;
TempCel = Convert(TempFer);
cout << "\nHere's the temperature in Celsius: ";
cout << TempCel << endl;
return 0;
}

float Convert(float Fer)
{
float Cel;
Cel = ((Fer - 32) * 5) / 9;
return Cel;
}
```

Tulosten pitäisi olla samoja.

Muuttujan näkyvyysalue (viittausalue), joka määrää, kuinka kauan muuttuja on käytettävissä ohjelmassa ja missä ohjelman osissa se on käytettävissä. Lohkon sisällä olevat muuttujat ovat näkyvissä vain lohkon sisällä; niitä voidaan käyttää vain lohkon sisällä eivätkä ne näy muualle. Globaalit muuttujat ovat näkyvillä kaikkialla ohjelmassa.

Yleensä näkyvyysalue on helposti ymmärrettävä asia, mutta on olemassa joitakin hankalia poikkeuksia. Nykyisin for-silmukan otsikossa esitellyt muuttujat (for int i = 0; i<SomeValue; i++) ovat näkyvillä lohkossa, jossa for-silmukka luotiin, mutta viime aikoina on puhuttu standardin muuttamisesta.

Näkyvyysalue ei tuottane hankaluuksia, kunhan vain olet huolellinen.

Globaalit muuttujat

Uusi käsite Funktion ulkopuolella esitellyillä muuttujilla on globaali näkyvyysalue ja siten mikä tahansa ohjelman funktio, mukaan lukien main(), voi käsitellä niitä.

C++ -kielessä vältetään globaalien muuttujien käyttämistä, koska ne luovat sekavaa koodia, jota on vaikea ylläpitää. Tämän kirjan esimerkeissä ei

käytetä lainkaan globaaleja muuttujia, enkä käytä niitä muulloinkaan ohjelmissani.

Funktion runko

Funktion rungossa voi olla rajaton määrä eri tyyppisiä ohjelmalauseita.

Toisaalta hyvin suunnitellut funktiot ovat yleensä lyhyitä. Suurin osa funktioista sisältää vain kourallisen koodirivejä.

Funktion argumentit

Funktion argumenttien ei tarvitse olla samaa tyyppiä. On täysin järkevää kirjoittaa funktio, joka ottaa kokonaisluvun, kaksi long-tyyppiä ja merkin argumenteikseen.

Mikä tahansa kelvollinen ilmaus voi olla funktion argumenttina, mukaan lukien vakiot, matemaattiset ja loogiset ilmaukset tai toiset, arvoja palauttavat funktiot.

Funktioiden käyttö toisten funktioiden parametreina

Vaikkakin on täysin hyväksyttävää käyttää arvon palauttavaa funktiota toisen funktion argumenttina, tekee menettely koodista vaikealukuista ja virhealtista.

Olettakaamme, että meillä on neljä funktiota double(), triple(), square() ja cube(), jotka kukin palauttavat arvon. Voisimme nyt kirjoittaa

```
Answer = (double(triple(square(cube(myValue)))));
```

Lause vie muuttujan myValue argumenttina funktiolle cube(), jonka palauttama arvo menee funktiolle square(), jonka palauttama arvo taas viedään funktiolle triple(). Lopuksi funktio double() saa triple()-funktion palauttaman arvon ja lopullinen arvo sijoitetaan muuttujaan Answer.

Tällainen koodi ei ole kovinkaan helppolukuista (oliko arvo kerrottu kolmella ennen neliöintiä?) ja jos lopputulos on virheellinen, niin mistä funktiosta virhe johtuu.

Vaihtoehtona on suorittaa koodi vaiheittain käyttäen apumuuttujia:

```
unsigned long myValue = 2;
unsigned long cubed = cube(myValue();  // tulos 8
unsigned long squared = square(cubed);  // tulos 64
unsigned long tripled = triple(squared); // tulos 192
unsigned long Answer = double(tripled);  // tulos 384
```

Nyt kutakin välitulosta on mahdollista tutkia ja suoritusjärjestys on selvä.

Parametrit ovat paikallisia muuttujia

Funktiolle viedyt parametrit ovat paikallisia funktiolle. Argumentteihin tehdyt muutokset eivät vaikuta kutsuvan funktion arvoihin. Menettelyä kutsutaan arvon viemiseksi, mikä tarkoittaa sitä, että funktiossa tehdään paikallinen kopio kustakin argumentista. Näitä paikallisia kopioita kohdellaan aivan kuin mitä tahansa muita paikallisia muuttujia. Listaus 5.3 havainnollistaa tätä.

Listaus 5.3. Arvon vieminen.

```
1:  // Listaus 5.3 - arvon 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 x, int y)
18: {
19:   int temp;
20:
21:   cout << "Swap. Before swap, x: " << x << " y: " << y << "\n";
22:
23:   temp = x;
24:   x = y;
25:   y = temp;
26:
27:   cout << "Swap. After swap, x: " << x << " y: " << y << "\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: 5 y: 10
```

Analyysi

Ohjelma alustaa kaksi muuttujaa main()-funktiossa ja vie ne sitten swap()-funktiolle, joka vaihtaa ne. Kun ne on tutkittu uudelleen main()-funktiossa, nähdään niiden olevan kuitenkin muuttumattomia!

Muuttujat alustetaan rivillä 9 ja niiden arvot esitetään rivillä 11. Sitten kutsutaan swap()-funktiota, jolle viedään muuttujat.

Rivillä 21 tulostetaan arvot uudelleen. Ne ovat samassa järjestyksessä kuin main()-funktiossa. Riveillä 23-25 vaihdetaan muuttujien paikkaa ja tuo vaihto vahvistetaan tulostusrivillä 27. Nähdään, että swap()-funktion sisällä ovat arvot vaihtuneet.

Suoritus palaa sitten riville 13 eli takaisin main()-funktioon, jossa arvot ovat alkuperäisessä järjestyksessä.

Kuten olet huomannut, swap()-funktiolle viedään argumentit arvoina, jolloin tehdään paikalliset kopiot muuttujista. Nuo paikalliset muuttujat vaihdetaan riveillä 23-25, mutta main()-funktiossa eivät muutokset näy.

Myöhemmin tutkimme vaihtoehtoisia tapoja viedä tietoa funktioihin ja näet, että funktiot voivat muuttaa pysyvästi niille vietyjen argumenttien arvoja.

Palautusarvot

Funktiot palauttavat joko arvon tai voidin, joka kertoo kääntäjälle, että mitään arvoa ei palauteta.

Funktio palauttaa arvon komennolla return, jota seuraa palautettava arvo. Arvo voi olla itsekin arvon palauttava ilmaus, kuten:

```
Return 5;
Return (x > 5);
Return (MyFunction());
```

Kaikki edellä olevat lauseet ovat hyväksyttyjä olettaen, että MyFunction() itse palauttaa arvon. Toisen lauseen, return (x > 5), arvo on nolla, jos x on suurempi kuin 5, muutoin se on yksi. Lause ei siis palauta x:ää vaan vertailulauseen arvon.

Kun return-lause kohdataan, palautetaan return-lauseen jälkeen oleva ilmaus funktion arvona. Ohjelman suoritus palaa kutsuvaan funktioon, kutsua seuraavalle riville.

Yhdessä funktiossa voi olla useampia kuin yksi return-lause. Muista kuitenkin, että heti, kun jokin return-lause suoritetaan, funktio päättyy. Listaus 5.4 havainnollistaa tätä ideaa.

Listaus 5.4. Useita return-lauseita.

```
1:  // Listaus 5.4 - palautetaan useita arvoja
2:  // return-lauseissa
3:
4:  #include <iostream.h>
5:
6:  int Doubler(int AmountToDouble);
7:
8:  int main()
9:  {
```

```
10:
11:    int result = 0;
12:    int input;
13:
14:    cout << "Enter a number between 0 and 10,000 to double: ";
15:    cin >> input;
16:
17:    cout << "\nBefore doubler is called... ";
18:    cout << "\ninput: " << input << " doubled: " << result << "\n";
19:
20:    result = Doubler(input);
21:
22:    cout << "\nBack from Doubler...\n";
23:    cout << "\ninput: " << input << "   doubled: " << result << "\n";
24:
25:
26:    return 0;
27: }
28:
29: int Doubler(int original)
30: {
31:    if (original <= 10000)
32:      return original * 2;
33:    else
34:      return -1;
35:    cout << "You can't get here!\n";
36: }
```

Tulostus

```
Enter a number between 0 and 10,000 to double: 9000
Before double is called…
Input: 9000 double: 0
Back from doubler…
Input: 9000 doubled: 18000

Enter a number between 0 and 10,000 to double: 11000
Before double is called…
Input: 11000 double: 0
Back from doubler…
Input: 11000 doubled: -1
```

Analyysi

Riveillä 14-15 kehotetaan antamaan luku, joka sitten tulostetaan rivillä 18 yhdessä paikallisen muuttujan, result, kanssa. Funktiota Doubler() kutsutaan rivillä 20 ja syötetty arvo viedään funktiolle parametrina. Tulos sijoitetaan paikalliseen muuttujaan result ja arvot tulostetaan uudelleen riveillä 22-23.

Rivillä 30, funktiossa Doubler(), testataan parametri, jotta nähdään, onko arvo suurempi kuin 10,000. Jos se ei ole suurempi, funktio palauttaa kaksi kertaa alkuperäistä arvoa suuremman arvon. Jos arvo on suurempi kuin 10,000, funktio palauttaa arvon -1, joka toimii virhearvona.

Rivin 35 lausetta ei koskaan suoriteta, koska funktion return-lause kohdataan ennen sitä. Hyvä kääntäjä varoittaa siitä, että ko. lausetta ei suoriteta ja hyvä ohjelmoija poistaa tuollaiset lauseet. Jotkut kääntäjät generoivat jopa virheen tuon rivin takia. Jos saat virheen, kommentoi tuo rivi 35 pois.

Oletusparametrit

Jokaisen prototyypissä ja määrittelyssä olevan paremetrin paikalla tulee olla arvo funktiota kutsuttaessa. Tuon arvon tulee olla myös esiteltyä tyyppiä. Niinpä, jos funktion esittely on seuraavanlainen:

```
long myFuntion(int);
```

täytyy funktiolle viedä kokonaislukumuuttuja. Jos funktion määrittely on erilainen tai jos funktiolle ei viedä kokonaislukua, saadaan kääntäjän virhe.

Ainoa poikkeus on silloin, kun funktion prototyyppi esittelee parametrin oletusarvon. Oletusarvo on arvo, jota käytetään silloin, kun mitään arvoa ei viedä. Edellä oleva esittelylause olisi voitu kirjoittaa näin:

```
long myFunction (int x = 50);
```

Tuo esittely kertoo, että "myFunction palautta long-tyypin ja kokonaislukuparametrin. Jos argumenttia ei viedä, käytetään oletusarvoa 50." Koska funktion prototyypissä ei vaadita parametrin nimeä, olisi esittely voitu kirjoittaa myös näin:

```
long myFunction (int = 50);
```

Funktion määrittely ei muutu oletusparametrin takia. Määrittelyn otsikko-osa olisi siis:

```
long myFunction (int x);
```

Jos kutsussa ei käytetä parametria, täyttää kääntäjä x:n oletusarvolla 50. Oletusparametrin nimen ei tarvitse olla sama esittelyssä ja määrittelyssä; oletusarvo sijoitetaan sijainnin, ei nimen perusteella.

Mikä tahansa tai kaikki funktion parametrit voivat saada oletusarvot. Ainoa rajoitus on seuraava: Jos jollakin funktion parametrilla ei ole oletusarvoa, ei oletusarvoa saa olla myöskään millään aiemmalla parametrilla.

Jos funktion prototyyppi on seuraavanlainen:

```
long myFunction ( int Param1, int Param2, int Param3);
```

voit sijoittaa oletusarvon Param2-parametriin vain, jos parametrilla Param3 on oletusarvo. Voit sijoittaa oletusarvon parametriin Param1 vain, jos parametreilla Param2 ja Param3 on oletusarvo. Listaus 5.5 esittelee oletusarvojen käyttöä.

Listaus 5.5. Parametrien oletusarvot.

```
1:  // Listaus 5.5 - esittelee oletusarvojen
2:  // käyttöä
3:
4:  #include <iostream.h>
5:
6:  int AreaCube(int length, int width = 25, int height = 1);
7:
8:  int main()
9:  {
10:   int length = 100;
11:   int width = 50;
12:   int height = 2;
13:   int area;
14:
15:   area = AreaCube(length, width, height);
16:   cout << "First area equals: " << area << "\n";
17:
18:   area = AreaCube(length, width);
19:   cout << "Second time area equals: " << area << "\n";
20:
21:   area = AreaCube(length);
22:   cout << "Third time area equals: " << area << "\n";
23:   return 0;
24: }
25:
26: AreaCube(int length, int width, int height)
27: {
28:
29:   return (length * width * height);
30: }
```

Tulostus

```
First area equals: 10000
Second time area equals: 5000
Third time area equals: 2500
```

Analyysi

Rivi 6 kertoo, että AreaCube() ottaa kolme kokonaislukuparametria, joista kahdella viimeisellä on oletusarvot.

Funktio laskee kuution parametreina vietyjen mittojen mukaan. Jos width-arvoa ei viedä, on oletuksena 25 ja height-arvona 1. Jos width-arvo annetaan, mutta ei height-arvoa, on height-oletuksena 1. Funktiolle ei voida viedä height-arvoa viemättä myös width-arvo.

Riveillä 10-12 alustetaan muuttujat length, height ja width ja ne viedään funktiolle AreaCube(). Tuloksen laskemisen jälkeen suoritetaan tulostaminen rivillä 16.

Suoritus palaa riville 18, jossa funktiota kutsutaan uudelleen, nyt ilman height-arvoa. Tällöin käytetään oletusarvoa.

Funktiota kutsutaan vielä uudelleen, nyt ilman width- ja height-arvoja.

Funktioiden ylikuormitus

C++ mahdollistaa useamman kuin yhden samannimisen funktion luomisen. Sitä kutsutaan funktion ylikuormittamiseksi. Funktioiden tulee olla erilaisia parametriluetteloiden suhteen, eli parametrien tyypit tai niiden määrät voivat vaihdella. Seuraavassa on esimerkki:

```
int myFunction (int, int);
int myFunction (long, long);
int myFunction (long);
```

Funktio on ylikuormitettu kolmella erilaisella parametriluettelolla. Ensimmäinen ja toinen esittely eroavat toisistaan parametrien tyyppien kohdalla ja kolmas esittely poikkeaa parametrien määrän suhteen.

Palautustyypit voivat olla samoja tai erilaisia ylikuormitetuilla funktioilla, kunhan vain parametriluettelot poikkeavat toisistaan. Ylikuormittamista ei voida kuitenkaan toteuttaa pelkästään palautustyyppien eroilla.

Funktioiden ylikuormittamista kutsutaan myös polymorfiaksi (polymorphism) eli monimuotoisuudeksi. Poly tarkoittaa monta ja morph muotoa.

Funktion monimuotoisuus viittaa mahdollisuuteen ylikuormittaa funktio useassa eri merkityksessä. Muuttamalla parametrien tyyppejä tai lukumäärää voidaan usealle funktiolle antaa sama nimi, jolloin oikea funktio tulee kutsutuksi parametrien täsmätessä. Menettely mahdollistaa sellaisen funktion luomisen, joka voi laskea keskiarvoja kokonaisluvuista, double-arvoista sekä muista arvoista tarvitsematta luoda useaa eri nimistä funktiota.

Olettakaamme, että kirjoitat funktion, joka kaksinkertaistaa minkä tahansa antamasi lukuarvon. Funktiolle tulee viedä parametrina int-, long-, float- tai double-tyyppinen arvo. Ilman ylikuormittamista olisi luotava neljä eri funktiota:

```
int DoubleInt(int);
long DoubleLong(long);
float DoubleFloat(float);
double DoubleDouble(double);
```

Ylikuormittamista käyttäen saataisiin sama aikaan seuraavasti:

```
int Double(int);
long Double(long);
float Double(float);
double Double(double);
```

Nyt koodi on luettavampaa ja funktioita on helpompi käyttää. Oikea funktio tulee kutsutuksi sille viedyn parametrin mukaan.

Inline-funktiot

Normaalisti funktio määriteltäessä kääntäjä luo yhden komentojoukon muistiin. Kun funktiota kutsutaan, suoritus hyppää tuohon komentojoukkoon ja funktion päättyessä suoritus palaa kutsua seuraavalle riville. Jos funktiota kutsutaan 10 kertaa, ohjelma siirtyy suorittamaan samaa koodijoukkoa joka kerta. Eli muistissa on vain yksi koodijoukko, ei 10.

Hypyt aiheuttavat hieman tehottomuutta suoritukseen. Jotkut funktiot ovat hyvin pieniä, ehkä vain pari koodiriviä, ja ohjelmaa voitaisiin tehostaa, jos siirtymisiä voitaisiin vähentää. Yleensä tehokkuuden merkkinä on nopeus, joka paranisi, jos funktioiden kutsuja voitaisiin vähentää.

Jos funktio esitellään lisämääreellä inline, kääntäjä ei luo todellista funktiota: se kopioi koodin inline-funktiosta suoraan kutsuvaan funktioon. Mitään hyppyjä ei tapahdu, vaan kutsuttavan funktion koodi on ikään kuin kirjoitettuna kutsuvaan funktioon.

Kuitenkin myös inline-funktioilla on heikot puolensa. Jos funktiota kutsutaan 10 kertaa, kopioidaan inline-koodi joka kerta kutsuviin funktioihin. Osa aikaansaadusta tehokkuudesta katoaa suoritettavan ohjelman koon kasvuun. Nopeuden parannuskin on hieman kuvitteellinen. Ensinnäkin modernit kääntäjät pyrkivät tehostamaan koodia varsin hyvin ja toiseksi inline-funktioita tulee kirjoitetuksi hyvin vähän. On tärkeää tietää, että koon kasvu vähentää tehokkuutta sekin.

Milloin inline-funktiot sitten sopisivat käyttöön? Aina, kun eteen tulee pieniä funktioita (pari koodiriviä), tulisi harkita inline-funktioiden käyttöä. Jos tehokkuuden paranemista epäillään, on parempi jättää inline-menettely pois. Listaus 5.6 esittelee inline-funktioiden käyttöä.

Listaus 5.6. Esimerkki inline-funktiosta.

```
1:
2:
3:  #include <iostream.h>
4:
5:  inline int Double(int);
6:
7:  int main()
8:  {
```

```
9:    int target;
10:
11:   cout << "Enter a number to work with: ";
12:   cin >> target;
13:   cout << "\n";
14:
15:   target = Double(target);
16:   cout << "Target: " << target << endl;
17:
18:   target = Double(target);
19:   cout << "Target: " << target << endl;
20:
21:
22:   target = Double(target);
23:   cout << "Target: " << target << endl;
24:   return 0;
25: }
26:
27: int Double(int target)
28: {
29:   return 2 * target;
30: }
```

Tulostus

```
Enter a number to work with:  20
Target: 40
Target: 80
Target: 160
```

Analyysi

Rivillä 5 esitellään Double() inline-funktiona, joka ottaa int-parametrin ja palauttaa int-tyypin. Esittely on aivan samanlainen kuin normaalisti lukuunottamatta lisämäärettä inline.

Lisämääreen kautta funktio käännetään aivan kuin olisit kirjoittanut

```
target = 2*target;
```

joka paikkaan, jossa on koodi

```
target = Double(target);
```

Ohjelmaa ajettaessa ovat lauseet jo paikoillaan, käännettynä .OBJ-tiedostoon. Näin suoritusaikaa ei käytetä funktiokutsuihin, mutta toisaalta ohjelman koko kasvaa.

Kuinka funktiot toimivat - katse pintaa syvemmälle

Kun funktiota kutsutaan, koodi haarautuu kutsuttavaan funktioon, parametrit viedään funktiolle ja funktion runko suoritetaan. Kun funktio

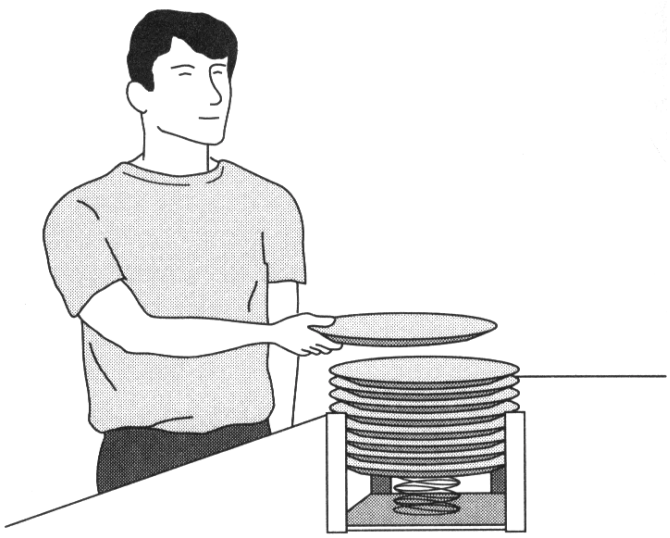
päättyy, palautetaan arvo (ellei funktio ole void-tyyppinen) ja suoritus palaa kutsuvalle funktiolle.

Kuinka nuo tehtävät toteutetaan? Kuinka tiedetään, minne tulee haarautua? Missä muuttujat pidetään niiden viemisen jälkeen? Mitä tapahtuu funktion rungossa esitellyille muuttujille? Kuinka palautusarvo viedään? Kuinka paluu osataan tehdä oikeaan paikkaan?

Pino

Uusi käsite

Kun ohjelma ajetaan, kääntäjä luo pinon. Pino on erikoinen muistialue, jonka ohjelma varaa funktioiden tarvitsemaa tietoa varten. Muistialuetta kutsutaan pinoksi, koska se toimii LIFO-periaatteella (Last In First Out, viimeksi laitettu tieto otetaan ensimmäisenä ulos). Ajattele pinoa vaikkapa astiapinona, jota kuva 5.4 esittää.



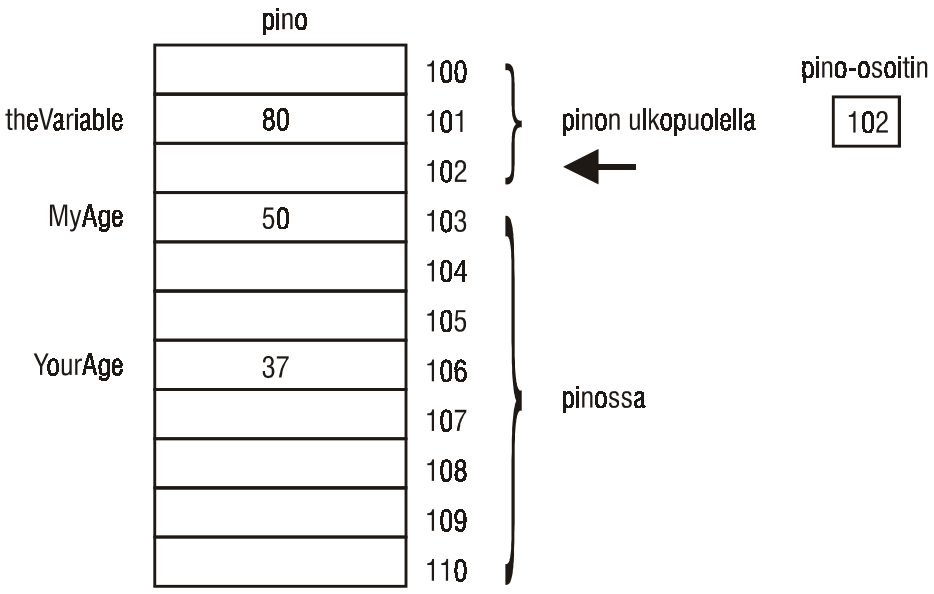
Kuva 5.4. Pino

Jono on toisenlainen muistialue, jossa ensimmäiseksi tullutta myös palvellaan ensimmäiseksi. Jonossa uusi alkio tulee aina loppupäähän (häntäpäähän) ja jono purkautuu jonon alkupäästä. Pinoon taas tulee uusi alkio pinon alkuun, päälle (push-toiminto) ja purkautuu päältä (pop-toiminto).

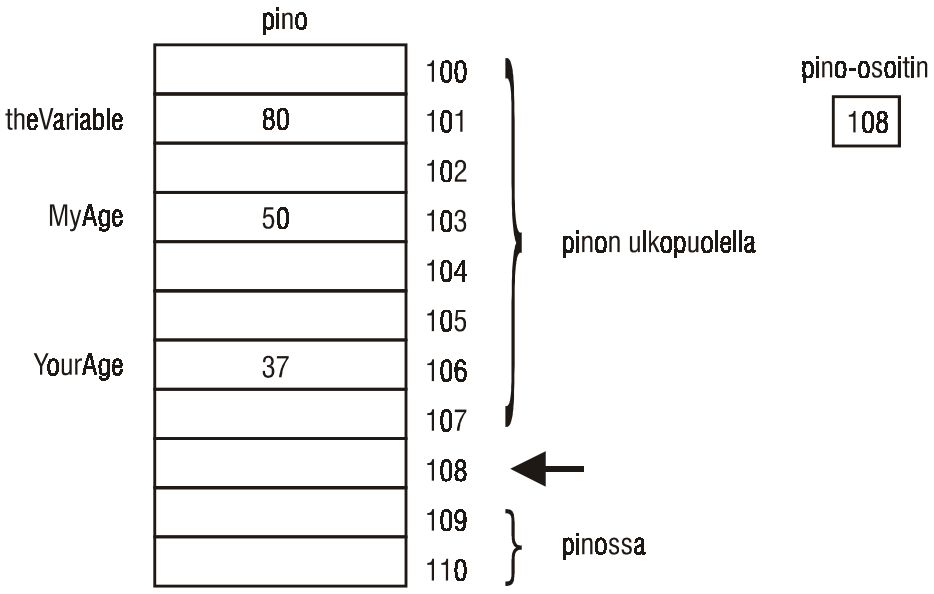
Pinon alkuun osoittaa muistissa pino-osoitin, jossa on tallessa pinon alun osoite. Koko pino sijaitsee sitten tuon pino-osoittimen kertoman osoitteen alapuolella. Kuva 5.5 havainnollistaa pinon sijaintia muistissa.

Kun tietoa laitetaan pinoon, se sijoitetaan pino-osoittimen yläpuolelle, jonka jälkeen pino-osoitinta kasvatetaan, jotta se osoittaisi uuteen tietoon. Kun

tietoa otetaan pinosta, muutetaan pino-osoittimen osoitetta siirtämällä osoitinta alaspäin. Kuva 5.6 havainnollistaa tätä menettelyä.



Kuva 5.5. Pino-osoitin.



Kuva 5.6. Pino-osoittimen siirtäminen.

Pino ja funktiot

Kun ohjelmasi kutsuu funktiota, toteutetaan pinokehys. Pinokehys on pinoalue, joka varataan funktion hallintaa varten. Hallintamenettely vaihtelee hieman eri tietokoneissa, mutta päävaiheet ovat seuraavat:

- ❏ Funktion paluuosoite sijoitetaan pinoon. Kun funktio päättyy, saadaan täältä paluuosoite.
- ❏ Varataan tilaa jonosta funktion palautustyypin mukaisesti.
- ❏ Kaikki argumentit sijoitetaan pinoon.
- ❏ Ohjelma haarautuu suorittamaan funktiota.
- ❏ Paikalliset muuttujat sijoitetaan pinoon niiden määrittelyjärjestyksessä.

Yhteenveto

Tässä luvussa käsiteltiin funktioita. Funktio on aliohjelma, jolle voidaan viedä parametreja ja joka voi palauttaa arvon. Jokainen C++ -ohjelma alkaa main()-funktiolla, joka taas kutsuu muita funktioita.

Funktio esittely on funktion prototyyppi, joka määrittää funktion palautusarvon, nimen ja parametrit tyyppeineen. Funktio voidaan esitellä inline-funktiona. Funktion prototyyppi voi myös esitellä oletusarvot yhdelle tai usealle parametrilleen.

Funktion määrittelyn tulee vastata funktion esittelyä palautustyypin, nimen ja parametriluettelon suhteen. Funktion nimeä voidaan ylikuormittaa muuttamalla parametrien tyyppiä tai määrää; kääntäjä löytää oikean funktion funktiolle vietyjen parametrien mukaan.

Paikalliset muuttujat ja funktiolle viedyt argumentit ovat paikallisia siinä lohkossa, jossa ne on esitelty. Arvoina viedyt parametrit ovat kopioita eikä niitä käsittelemällä voida muuttaa kutsuvan funktion arvoja.

Kysymyksiä ja Vastauksia

K
Miksei käytetä pelkästään globaaleja muuttujia?
V
Aikoinaan niin juuri tehtiinkin. Ohjelmista tuli sitten laajempia ja monimutkaisempia ja virheiden hakeminen oli työlästä, koska mikä tahansa funktio oli voinut vaikuttaa muuttujien arvoihin. Kokeneet ohjelmoijat tietävät, että paikallisia muuttujia kannattaa käyttää niin paljon kuin mahdollista; tällöin tiedetään, mikä funktio on käsitellyt kutakin muuttujaa.

K
Miksi argumenttien muutokset eivät välity funktiosta kutsuvaan funktioon?
V
Funktiolle viedään argumentit arvoina (normaalisti). Tällöin funktio saa kopion arvoista eikä siis alkuperäisiä arvoja.

K
Mitä tapahtuu, jos käytän seuraavia kahta funktiota?

```
int Area (int width, int length = 1);
int Area (int size);
```

Ovatko funktiot monimuotoisia? Funktioissa on eri määrä parametreja, mutta ensimmäisellä on oletusarvo.
V
Esittely käännetään, mutta jos käytät funktiota yhdellä parametrilla, saat virheilmoituksen: ambiguity between Area(int, int) and Area(int).
Eli menettelyssä on ristiriitaa.