TI02 Ohjelmointi

Kurssimoniste

Johdanto ja merkinnöistä

Tämän monisteen tarkoituksena on antaa riittävät perustiedot C++-ohjelmoinnin opiskelun aloittamiseen. Käsitellään yksinkertaisia ohjelmia ja kiinnitetään erityistä huomiota loogiseen ajatteluun ongelmanratkaisussa. Esitettävät ohjelmat on käännetty Windowsin C++ kääntäjällä, mutta ainakin teoriassa niiden pitäisi kääntyä myös Linuxilla yms. muilla käyttöjärjestelmillä.

Moniste seuraa erittäin tarkasti Bjarne Stroustrupin kirjaa

> **Programming: Principles and Practice Using C++** (Addison-Wesley 2008)

Kurssin tavoitteena on käydä soveltuvin osin läpi kirjan luvut 1–5, mutta jos opiskelijoilla on intoa, niin luku 6 sisältää mahdollisen lisäprojektin. Jokainen tämän monisteen luku sisältää yhden luvun Stroustrupin kirjasta tiivistettynä, alkaen kirjan luvusta 2. Monisteen luvut on nimetty kirjasta hieman poikkeavalla tavalla, sisältäen kuitenkin lähes kaikki tärkeimmät asiat.

Teksti on kirjoitettu LaTeX 2_ε ladontakielellä. Mikäli mielenkiintoa on, voi ohjelmiston ja peruskäytöstä kysellä allekirjoittaneelta lisää. LaTeX -koodia kääntävä ohjelma Windows-koneilla on **MikTeX**, jonka voi ladata netistä ilmaiseksi. Sopiva editori tekstin kirjoittamiseen on **TexnicCenter**.

Virolahdella 17. toukokuuta 2011,

Riku Järvinen

Sisältö

1 Hello, World!

Aloitetaan opiskelu kirjoittamalla yksinkertainen C++-ohjelma nimeltä `helloworld.cpp`, joka tulostaa komentoriville tekstin Hello World!

1.1 Ensimmäinen ohjelma

helloworld.cpp

```
1  // Ohjelma tulostaa ruudulle tekstin Hello World!
2
3  #include "../std_lib_facilities.h"
4
5  int main ()
6  {
7      cout << "Hello World!\n";
8      keep_window_open();
9      return 0;
10 }
```

Käydään koodi läpi rivi riviltä.

1. Merkintä `//` aloittaa yhden rivin kommentin. Yleensä ohjelman alussa halutaan kertoa, mitä ohjelman olisi tarkoitus tehdä (ei tarkkaa selostusta, vain perusidea).

3. Omaan koodiin yhdistetään `#include`-direktiivillä C++:n standardikirjaston tiedot. Standardikirjastossa on määritetty mm. riviltä 7 löytyvät funktio `cout`, joka tulostaa tekstiä ruudulle. Merkintä `../` tarkoittaa, että haettu tiedosto `std_lib_facilities.h` on yhtä hakemistotasoa ylempänä kuin tiedosto `helloworld.cpp`. Tiedostopääte `.h` puolestaan tarkoittaa ns. **header-tiedostoa** ja `.cpp` itse kirjoitettua muokattavaa **lähdekooditiedostoa**.

5. `int main` aloittaa varsinaisen ohjelman suorittamisen. Tavalliset sulut `()` jätetään tyhjiksi. Varsinainen ohjelmakoodi tulee aaltosulkujen `{...}` sisään eli **funktiorunkoon**.

6. Aaltosulku aloittaa funktiorungon ja on tavallisesti omalla rivillään tai edellisen rivin lopussa.

7. Standardikirjaston funktio `cout` ja **output-operaattori** `<<` tulostavat komentoriville. Tavallinen, kirjoitettava teksti on lainausmerkkien sisällä (tulostetaan juuri sellaisena kuin on, myös tyhjät välit) ja riviä vaihdetaan kirjoittamalla `\n` lainausmerkkien sisään. Lauseke

   ```
   cout << "Hello, World!\n";
   ```

   päättyy puolipisteeseen, kuten kaikki lausekkeet C++:ssa.

8. Funktio `keep_window_open()` pitää ohjelmaikkunan auki: kätevä, sillä muuten ikkuna sulkeutuisi välittömästi ohjelman ajamisen jälkeen.

9. Lauseke `return 0;` ei varsinaisesti tee mitään, vaan ainoastaan esittelee sitä että funktio voi palauttaa jonkin arvon tarvittaessa (vain yhden arvon). Arvoa 0 käytetään joskus tarkistamaan, että ohjelma on tehnyt sen, mitä pitääkin (Linux/Unix-järjestelmissä).

10. Aaltosulku lopettaa main-funktion eli koko ohjelman suorittamisen.

1.2 Funktio

C++:n funktio on määritelty aivan kuten matematiikassa, eli sille voidaan antaa yksi tai useampi argumentti ja se palauttaa täsmälleen yhden arvon. Funktiolla on aina *nimi*, *palautusarvon tyyppi*, *parametrilista* (argumentit) ja *funktion runko*, joka sisältää varsinaisen toiminnallisuuden. Funktioilla voidaan automatisoida usein toistuvia toimia, jotka muuten vaatisivat pitkän saman koodin kirjoittamista joka kerta erikseen. Funktioiden käyttö selventää koodia loogisesti, sillä rutiinitoiminnot on kuitattu useimmiten vain yhdellä rivillä.

1.3 Koodin kääntäminen ja linkittäminen ohjelmaksi

Kirjoitettava koodi eli **lähdekoodi** pitää muuttaa eli **kääntää** tietokoneen ymmärtämään muotoon **konekielelle** (machine code tai object code). Työn tekee **kääntäjäohjelma** (compiler), joka

- lukee lähdekoodin `// ilman siinä olevia kommentteja`
- tarkastaa kieliopin oikeellisuuden
- ilmoittaa virheestä. Jos virhe löytyy, niin käännöstyötä ei tehdä. Tällöin lähdekoodin virhe tulee korjata ja kääntää koodi uudelleen. Tavallisimpia virheitä ovat mm. lainausmerkin, sulkumerkin tms. puuttuminen, sanojen unohtaminen, väärien muuttujien nimien käyttö, väärien operaattorien käyttö ja puolipisteen puuttuminen suoritettavan lauseen lopusta.

Lyhyesti:

Kääntäjä on yleensä aina oikeassa, ihmiset väärässä.

Kääntäjä on ohjelmoijan paras ystävä koodatessa. Kokemuksen kautta oppii, että ilman kääntäjää menisi erittäin paljon turhaa aikaa yksinkertaisten kirjoitusvirheiden korjaamiseen.

C++ ohjelma koostuu tavallisesti useasta eri osasta. `helloworld.cpp` käyttää omaa lähdekoodiaan ja lisäksi C++:n standardikirjaston osia. Itse kirjoitettu ja kääntäjällä käännetty lähdekoodi linkitetään standardikirjaston lähdekoodiin, ja linkitys yhdistää nämä koodit suoritettavaksi tiedostoksi (.exe-tiedosto). Lähdekoodista saadaan .exe-tiedosto kirjoittamalla komentoriville

```
g++ helloworld.cpp -o helloworld.exe
```

Käännöstyön suorittaa ohjelma nimeltä **g++**, joka on osa **GCC**:tä (*GNU Compiler Collection*). Argumentit kannattaa kirjoittaa oikeassa järjestyksessä, sillä esimerkiksi komento

```
g++ -o tiedosto.cpp ohjelma.exe
```

tuhoaa alkuperäisen lähdekooditiedoston pysyvästi eikä käännä ohjelmaa. Sinua on varoitettu...

> **HUOMIO**. Konekielinen koodi ja suoritettava ohjelma eivät ole alustariippumattomia, eli Windowsilla käännetty ja linkitetty ohjelma ei toimi GNU/Linux-koneella.

1.4 C++:n standardikirjasto

Linkkeri yhdistää oman lähdekoodin ja standardikirjaston koodin yhdeksi toimivaksi ohjelmaksi. Standardikirjasto sisältää yksinkertaisesti koodia, jossa määritellään kielen peruskomponentteja, mm. ruudulle tulostaminen ja käyttäjän syötteen lukeminen sekä tiedon tallennukseen liittyviä seikkoja. Standardikirjastokoodin voi ladata esimerkiksi Bjarne Stroustrupin henkilökohtaiselta sivulta

`http://www.stroustrup.com/`

Tämä versio voi olla vanhentunut, mutta on tehty nimenomaan kurssikirjaa varten joten toimii riittävän hyvin sen kanssa. Standardikirjasto linkitetään omaan koodiin `#include`-direktiivillä (tarkoittaa käytännössä samaa kuin että koko kirjaston koodi olisi kopioitu oman koodin alkuun. Tätä halutaan välttää, sillä tekniset yksityiskohdat eivät ole kiinnostavia tässä vaiheessa).

1.5 Virhetyypit sekä ohjelmointi käytännössä

Ohjelmointityössä törmää seuraavanlaisiin virheisiin:

- kääntöajan virheet (compile-time errors)
- linkitysajan virheet (link-time errors)
- suorituksenaikaiset virheet (run-time errors)

Pääsääntönä on, että kääntöajan virheet on helpointa korjata, suorituksen aikaiset virheet työläintä. Virheitä tarkastelemme jonkin verran kurssin loppupuolella luvussa 5.

Ohjelmointia voi tehdä joko käyttäen **komentoriviä** ja **tekstieditoria** tai **kehitysympäristöä** (IDE, Integrated Development Environment). IDE:n etuja ovat usein monipuoliset koodin käsittelyominaisuudet, haittoja raskas käyttöliittymä (liian paljon ominaisuuksia) ja ei-niin-suora koodin suorittaminen kuin komentoriviltä. Ohjelmoinnin alussa kannattaa käyttää komentoriviä ja sitten halutessaan voi siirtyä IDEn käyttöön jossakin vaiheessa, kunhan ymmärtää ensin perusasiat.

1.6 Vinkkejä: komentorivin käyttö

Ohjelmointi suoritetaan tällä kurssilla kokonaan komentoriviltä, sillä samalla se opettaa tuntemaan tietokoneen rakennetta hieman paremmin. Komentorivin saa auki painamalla Win-R, kirjoittamalla *Suorita*-ikkunaan tekstin *cmd* ja painamalla *Enter*. Kansioissa voi navigoida taaksepäin komennolla `cd ..` ja eteenpäin komennolla `cd hakemiston_nimi`, esimerkiksi `cd Omat Tiedostot`. Tabulaattori täydentää sanat, eli voit kirjoittaa vain `cd Om` ja painaa Tabia, jolloin nimi täydentyy ja Enterillä voit siirtyä ks. hakemistoon.

Komentoriviltä voi suoraan kopioida tekstiä, esimerkiksi kääntäjän virheilmoituksia. Painetaan ALT-SPACE → ominaisuudet → pikamuokkaustila. Nyt voidaan valita tekstiä, kopioida sen painamalla Enteriä leikepöydälle ja liittää vaikkapa tekstitiedostoon. Tämä on hyödyllistä esimerkiksi silloin, kun haluaa analysoida virhelogeja ja napata niistä osan talteen.

Lisätietoa komentorivin käytöstä löytyy osoitteesta `http://www.commandwindows.com`.

1.7 Kysymyksiä luvusta 1

1. Miillaisia työkaluja `helloworld.cpp` esittelee käyttöön?
2. Mikä on kääntäjä?
3. Mitä linkkeri tekee?
4. Mitä ovat .h-päätteiset tiedostot ja miksi niitä tarvitaan?
5. Mitä eroa on lähdekooditiedostolla ja konekielisellä tiedostolla?
6. Mitä etua/haittaa on komentorivin/IDEn käytöstä ohjelmointityössä?

1.8 Tehtäviä luvusta 1

1. Asenna koneelle tarvittavat ohjelmat ja hae Stroustrupin sivuilta standardikirjasto.
2. Kirjoita `helloworld.cpp` ja aja se komentoriviltä. Korjaa mahdolliset virheet, mikäli niitä ilmenee.
3. Kokeile, millä tavalla kääntäjä reagoi erilaisiin virheisiin. Testaa ainakin seuraavia:
   - unohtunut sulku funktiosta `main (`
   - unohtunut lainausmerkki standardikirjaston ympäriltä: `"std\_ lib\_ facilities.h`
   - pääohjelman `main` väärä nimi
   - pääohjelman väärä tyyppi, esimerkiksi `double main ()`
   - kirjoitusvirhe `cout` → `trout`
   - `"Hello World!\n` jälkimmäinen lainausmerkki jätetään pois
   - Unohdetaan puolipiste `"Hello World!\n"` lopusta
   - Rivinvaihto lainausmerkkien ulkopuolelle eli `"Hello World!"\n`

2 Objektit, tyypit ja arvot

Ohjelmointitaito vaatii kielen perusrakenteiden hyvää tuntemusta. Tämän luvun aikana tutuksi tulevat kielen yksinkertaisimmat osa eli mm. objektit, muuttujat, tyypit ja niihin liittyvät operaatiot.

Asioiden esittely voi tuntua ajoittain melko tekniseltä. Suosittelen, että lukiessa ensimmäisellä kerralla kannattaa edetä rauhallisesti siten, että ymmärtää käsitteiden merkityksen ja liittymisen toisiinsa mahdollisimman hyvin.

2.1 Käyttäjän syötteen lukeminen ja ruudulle tulostaminen

Luvussa 1 tulostimme tekstiä ruudulle kirjoittamalla `cout <<`. Käyttäjältä voidaan myös lukea tietoa, jolloin kirjoitetaan `cin >>`. Esimerkiksi kokonaisluku luetaan näin:

```
int kokonaisluku;
cin >> kokonaisluku; // cin = standard input stream
```

edellinen sijoittaa muuttujan `kokonaisluku` arvoksi sen, mitä käyttäjä kirjoittaa ruudulle. Nimen kirjoittamisen jälkeen painetaan *Enter*-näppäintä.

Joskus käyttäjältä pyydetään erityisesti jotain syötettä, esimerkiksi tulostamalla ruudulle seuraavanlainen teksti:

```
cout << "Kirjoita ruudulle, kuinka vanha olet.\n";
int oma_ika;
cin >> oma_ika;
```

Kyselystä käytetään erityistä nimeä **kehote** (*prompt*). Huomaa, että muuttujan `oma_ika` tulee olla määritelty ennen kuin siihen voidaan lukea arvo. Muuttujan **määritelmä** (*definition*) sisältää aina muuttujan tyypin ja nimen sekä mahdollisesti alkuarvon: esimerkiksi `int oma_ika = 10;` on järkevästi kirjoitettu lauseke, joka määrittelee muuttujan `oma_ika`.

Jos käyttäjältä halutaan lukea useita muuttujan arvoja, esimerkiksi kaksi kokonaislukua ja tulostaa ne sitten ruudulle, voidaan molemmat operaatiot kirjoittaa yhteen sisään- ja ulostulovirtaan:

```
int luku1;
int luku2;
cout << "Kirjoita kaksi lukua ruudulle\n";
cin >> luku1 >> luku2;
cout << luku1 << " " << luku2 << "\n";
```

Huomaa, että lainausmerkeissä oleva teksti tulostuu täsmälleen sellaisena kuin se on kirjoitettu. Lainausmerkkejä ei käytetä, kun tulostetaan jonkin muuttujan arvo.

2.2 Objektit ja tyypit: johdantoa

Tiedon tallennuspaikka tietokoneen muistissa on nimeltään **objekti** (*object*). Nimetty objekti on **muuttuja** (*variable*), jolle voidaan sijoittaa jokin **arvo** (*value*). **Lauseke** (*statement*), joka määrittelee muuttujan ja sen mahdollisen **alkuarvon** (**initial value**), on määritelmä. Esimerkkejä määritelmistä ovat ainakin edellä esitetyt `int luku1;` ja `int oma_ika = 10;`, joista jälkimmäinen määrää myös muuttujan alkuarvon.

Muuttujaan tulee sijoittaa sen **tyyppiä** vastaava arvo, sillä muuten kääntäjä antaa virheilmoituksen. Tyyppejä ovat mm. **double** (desimaaliluku eli **liukuluku**), **int** (kokonaisluku), **char** (merkki), **string** (merkkijono) ja **bool** (totuusarvo eli tosi tai epätosi). Lisäksi jokaisella tyypillä on oma **literaalinsa** eli muuttujan arvon merkitsemistapa, esimerkiksi

```
double b = 3.9;
int a = 39;
char c = 'c';
string d = "jotain tekstia"
bool totuus_arvo = true;
```

Jos muuttujalle antaa vääräntyyppisen arvon, niin arvoa luettaessa ruudulle voi tulostua täysin mielivaltainen arvo, esimerkiksi -9837. Kyseessä on se arvo, joka on tietokoneen muistissa ks. muistipaikassa oli täsmälleen sillä hetkellä, kun vääräntyyppistä arvoa yritettiin syöttää. Mikäli haluaa varmistua siitä, että oikeanlainen arvo on sijoitettu, niin kannattaa antaa muuttujalle sopiva alkuarvo (ikää kysyttäessä se voisi olla vaikkapa -1).

Merkkijonomuuttujan (string) lukeminen päättyy C++-kielessä aina välilyöntiin, rivinvaihtoon tai tabulaattorimerkkiin (*whitespace*). Luettaessa vaikkapa kaksiosaista nimeä tulee kirjoittaa

```
string etunimi;
string sukunimi;
cin >> etunimi >> sukunimi;
```

Huomaa, että syöteoperaattori >> reagoi tyyppiin, joten sille tulee antaa määritellyn muuttujan tyyppiä vastaava arvo, esimerkiksi seuraavasti:

```
cout << "Kirjoita ruudulle, kuinka vanha olet.\n";
int oma_ika;
cin >> oma_ika;
cout << "Olet " << oma_ika << " vuotta vanha\n";
```

Tarkastellaan nyt lyhyttä ohjelmaa, joka kysyy käyttäjältä etunimen ja iän.

2.2.1 Esimerkki: Käyttäjän ikä ja nimi

ikajanimi.cpp

```
1  #include "../std_lib_facilities.h"
2
3  int main()
4  {
5  cout << "Kirjoita etunimesi ja ik\x84si\n";
6  string etu_nimi;
7  double ika = -1.0; // alkuarvoksi "mahdoton arvo"
8  cin >> etu_nimi >> ika;
9  cout << "Hei, " << etu_nimi
10      << " (ik\x84 " << ika << " vuotta)";
11 }
```

Käydään koodia läpi rivi riviltä.

1. Otetaan standardikirjasto käyttöön yhtä hakemistotasoa ylempää.

5. Kirjoitetaan kehote, joka pyytää käyttäjää kirjoittamaan etunimen ja iän.

6. Määritellään merkkijonomuuttuja `etu_nimi`.

7. Määritellään desimaalilukumuuttuja `ika` ja alustetaan se siten, että voidaan tunnistaa myöhemmin mahdollinen virheellinen syöte (jos käyttäjä syöttää vääräntyyppisen arvon, niin ruudulle tulostuu -1 ja tiedämme, että syötetty arvo oli virheellinen).

8. Luetaan käyttäjältä `etu_nimi` ja `ika`.

9. Tulostetaan teksti `Hei, ...`, missä kolmen pisteen paikalla on käyttäjän kirjoittama nimi.

10. Tulostetaan `(ikä ... vuotta).`, missä kolmen pisteen paikalla on käyttäjän kirjoittama ikä. Huomaa, että kirjain `ä` on kirjoitettu lainausmerkkien sisään muodossa `x84`. Tämä on välttämätöntä, jotta se näkyy oikein (kääntäjä käyttää ilmeisesti eri merkistöä kuin Windows, joten skandit on kirjoitettava ns. *heksadesimaalimuodossa*. Vastaavasti kirjain `ö` kirjoitettaisiin `x94`).

2.3 Operaattorit

Muuttujan tyyppi määrittää, millaisia operaatioita muuttujaan voidaan kohdistaa. Kääntäjä antaa virheilmoituksen, jos tyyppi ja operaatio eivät sovi yhteen. Esimerkkejä operaatioista:

```
int numero;
string kirjaimet;
cin >> numero >> kirjaimet;
int numero_2 = numero + 1;
string kirjaimet_2 = kirjaimet + " ja jotain.";
cout << numero_2 << " " << kirjaimet_2;
```

`int`-tyypin muuttujaan voidaan lisätä lukuja ja `string`-tyypin muuttujaan voidaan kasvattaa, jolloin uusi merkkijono sisältää alkuperäisen lisäksi annetut uudet merkit. Operaatioiden ja tyyppien yhteensopivuudet oppii kokemuksen kautta ja alkuvaiheessa kannattaa katsoa asiaan liittyviä taulukoita. Valikoima operaattoreita on listattu taulukossa 1. Operaattoreiden toiminta saman kategorian muuttujille (esimerkiksi `int` ja `double`) on yleensä samankaltaista. Vaikka jokin operaatio ei suoraan olisi sallittu jollekin tyypille, niin voi olla epäsuora keino ks. operaation käyttämiseen. Katsotaan esimerkki operaatioiden käyttämisestä.

2.3.1 Esimerkki: Operaattoriharjoitus

operaattoriharjoitus.cpp

```
1  #include "../std_lib_facilities.h"
2
3  int main ()
4  {
5  cout << "Kirjoita ruudulle liukuluku\n";
6  double liukuluku;
7  cin >> liukuluku;
8  int liukuluku_toint = liukuluku; // tarvitaan moduloa varten
9  cout << "liukuluku == " << liukuluku
10      << "\n3 kertaa liukuluku == " << 3*liukuluku //rivinvaihto alussa!
11      << "\nliukuluku + liukuluku == " << liukuluku+liukuluku
12      << "\nliukuluvun neli\x94 == " << liukuluku*liukuluku
13      << "\nliukuluvusta puolet == " << liukuluku/2
```

TAULUKKO 1: Valikoima operaattoreita

Operaatio	**bool**	**char**	**int**	**double**	**string**
sijoitus (assignment)	=	=	=	=	=
lisäys / merkkijonon pidennys			+	+	+
vähennys			-	-	
kertominen			*	*	
jakaminen			/	/	
jakojäännös			%		
lisääminen yhdellä			++	++	
vähentäminen yhdellä			--	--	
lukee lähteestä `s` muuttujaan `x`	s » x	s » x	s » x	s » x	s » x
kirjoittaa arvon `x` ulostuloon `s`	s « x	s « x	s « x	s « x	s « x
yhtä suuri kuin	==	==	==	==	==
ei yhtä suuri kuin	!=	!=	!=	!=	!=
suurempi kuin	>	>	>	>	>
suurempi tai yhtä suuri kuin	>=	>=	>=	>=	>=
pienempi kuin	<	<	<	<	<
pienempi tai yhtä suuri kuin	<=	<=	<=	<=	<=

```
14      << "\nliukuluvun neli\x94juuri == " << sqrt(liukuluku)
15      <<"\nliukuluku modulo 2 on " << liukuluku_toint%2
16      << endl;
17  }
```

Ohjelmassa `operaattoriharjoitus.cpp` tarkastellaan liukuluvun eli desimaaliluvun (double) operaatioita. Katsotaan soveltuvin osin koodin tärkeimmät kohdat.

9. Liukuluku muutetaan kokonaisluvuksi eli määritellään kokonaislukumuuttuja `liukuluku_toint` ja annetaan sen arvoksi liukuluvun arvo. Tässä on kyseessä eräänlainen *tyyppimuunnos*, jossa tietoa häviää: desimaaliluku pyöristyy näissä tilanteissa aina alaspäin, esimerkiksi luku 1.9 arvoon 1. Kannattaa olla tarkkana, jos koodissa on tyyppimuunnoksia!

12. Skandikirjain `ö` on kirjoitettu heksadesimaalina muodossa `x94`.

14. Liukuluvun neliöjuuri saadaan käyttämällä hyväksi standardikirjaston funktiota `sqrt()`, joka ottaa argumenttina yhden liukuluvun. Funktion `sqrt()` sisäisestä rakenteesta ei tarvitse välittää tässä vaiheessa.

15. Modulo eli jakojäännös mittaa sitä, mitä jää jäljelle, kun kokonaisluvuksi muunnettu liukuluku jaetaan kahdella. Tuloksen pitäisi olla joko 0 tai 1 riippuen siitä, onko muuttujan `liukuluku_toint` arvo parillinen vai pariton.

16. Tulostusvirta `cout` lopetetaan ja rivi vaihdetaan kirjoittamalla tulostuksen loppuun `endl;`.

2.4 Nimet C++:ssa

Pääsääntöinä nimivalinnoissa muuttujille, funktioille tms. on, että

- nimi alkaa kirjaimella,
- sisältää vain kirjaimia (pienet ja isot), numeroita ja alaviivoja
- nimissä ei saa olla välilyöntejä.

Iso kirjain on eri asia kuin pieni kirjain ja skandeja ei käytetä nimissä. Noin 70 nimeä on varattu C++:ssa **avainsanoille** (*keywords*) ja niitä ei saa käyttää muuttujien, tyyppien, funktioiden ja vastaavien nimissä (esimerkkejä vaikkapa `int` ja `double`). Ei myöskään kannata käyttää standardikirjaston osien nimiä, kuten esim. `string` tai vastaavaa.

Nimen pitää olla kuvaava, sopivan pituinen ja mielellään sisältää pääosin pieniä kirjaimia. Esimerkkejä hyvistä nimistä ovat seuraavat:

```
tarkastaja_funktio
valuuttamuunnin
Elementtilaskuri
Toinen_vaihtoehto
```

Kannattaa välttää nimiä, jotka on helppo lukea tai kirjoittaa väärin tai sekoittaa toisiinsa (erityisesti kirjainten 0, O, o, 1, l ja I osalta). Katsotaan seuraavaksi esimerkki siitä, kuinka käy, jos nimen kirjoittaa väärin.

2.4.1 Esimerkki: virheilmoitukset nimissä

nimivirheilmoitukset.cpp

```
/* ============================================================================ */
/*                                                                              */
/*    nimivirheilmoitukset.cpp      */
/*    (c) 19.2.2011 Riku Järvinen*/
/*                                                                              */
/*    Description
      Katsotaan virheilmoituksia, kun muuttujien ja tyyppien nimissä on virheitä.
 ============================================================================ */

#include "../std_lib_facilities.h"

// laitetaan kommentteihin virheellinen koodi ja siihen liittyvä virhelogi,
// tähän alle puolestaan toimiva koodi.

int main()
{
    string s = "T\x84ss\x84 kokeillaan eri virhetyyppej\x84.";
    cout << s << '\n';
}

/*
Kirjoitetaan ohjelman ensimmäinen versio, joka sisältää paljon virheitä:

int Main()
{
    String s = "T\x84ss\x84 kokeillaan eri virhetyyppej\x84.";
    cOut << S << '\n';
}

Ylläoleva koodi tuo seuraavanlaisen virhelogin:

koodivirheilmoitukset.cpp: In function 'int Main()':
koodivirheilmoitukset.cpp:15:5: error: 'cOut' was not declared in this scope
koodivirheilmoitukset.cpp:15:13: error: 'S' was not declared in this scope

Eli kertoo, että virhe on funktiossa int Main(). cOut ei ole määritelty eikä S
ole määritelty. Korjataan virheet, eli laitetaan cout ensin oikein, ts.
seuraavan näköinen koodi:

int Main()
{
    String s = "T\x84ss\x84 kokeillaan eri virhetyyppej\x84.";
    cout << S << '\n';
}

Ylläoleva koodi tuo seuraavanlaisen virhelogin:
```

```
47
48 koodivirheilmoitukset.cpp: In function 'int Main()':
49 koodivirheilmoitukset.cpp:17:13: error: 'S' was not declared in this scope
50
51 Muutetaan String-muuttujan s nimi oikein eli korvataan S -> s.
52
53 int Main()
54 {
55     String s = "T\x84ss\x84 kokeillaan eri virhetyyppej\x84.";
56     cout << s << '\n';
57 }
58
59 Ylläoleva koodi tuo seuraavanlaisen virhelogin:
60
61 c:/mingw32/bin/../lib/gcc/mingw32/4.5.1/../../../libmingw32.a(main.o):main.c:(.t
62 ext+0xd2): undefined reference to `WinMain@16'
63 collect2: ld returned 1 exit status
64
65 Eli nyt kääntäjä ilmoittaa virheellisestä viitteestä Main-funktiossa: korjataan
66 Main -> main.
67
68 int main()
69 {
70     String s = "T\x84ss\x84 kokeillaan eri virhetyyppej\x84.";
71     cout << s << '\n';
72 }
73
74 Kääntäjä ei anna virheilmoitusta, tulostaa normaalisti. Näin ollen
75 String-muuttujan tyyppi (väärin kirjoitettu) ei ollut nähtävästi esteenä
76 ohjelman toiminnalle. Kirjoitetaan lopulliseen versioon kuitenkin oikeaoppinen
77 koodi, eli muutetaan String -> string.
78
79 */
```

2.5 Alkuarvon asettaminen ja sijoittaminen

Muuttujan **alkuarvon asettaminen** (*initialization*) tapahtuu operaattorilla =. Esimerkkejä ovat mm. seuraavat:

```
string merkkijono = "jotain merkkej\x84 t\x84ss\x84";
int luku = 32;
double isoluku = 3934783744*luku;
double viela_isompi_luku = sqrt(isoluku);
bool totuus = true;
```

Operaattorilla = voidaan myös **sijoittaa uusi arvo** (*assign*) jo olemassa olevaan muuttujaan:

```
a = "merkkijono"; // jo määritellyn string-muuttujan a arvoksi merkkijono
b = b+3; // lisätään b:n arvoon 3 ja sijoitetaan summa-arvo b:n paikalle
c = d%2 // olettaa, että c ja d ovat int-tyyppiä
```

Looginen ero sijoittamisen ja alkuarvon asettamisen välillä on se, että alkuarvon asettaminen ”löytää” muuttujan tyhjänä, kun taas sijoittaminen poistaa edellisen arvon ja laittaa tilalle uuden. Katsotaan esimerkkinä ohjelma, joka tarkastaa kirjoitetusta lauseesta, ovatko peräkkäiset sanat samoja.

2.5.1 Esimerkki: nimiharjoitus 1

nimiharjoitus.cpp

```
/* ========================================================================== */
/*                                                                            */
/*   nimiharjoitus.cpp                                                        */
/*   (c) 2011 Riku Järvinen                                                   */
/*                                                                            */
/*   Description                                                              */
/*   Tarkastaa kirjoitetusta lauseesta, ovat peräkkäiset sanat samoja.*/
/* ========================================================================== */

#include "../std_lib_facilities.h"

int main()
{
    cout << "Kirjoita teksti\x84, niin ohjelma etsii kahdentuneet sanat.\n";
    string edellinen = " ";  //edellinen sana: alkuarvo ei ole sana (fails 1st)
    string nykyinen;
    while (cin>>nykyinen) {
        if(edellinen == nykyinen) //onko edellinen saman kuin nykyinen
            cout << "Toistettu sana: " << nykyinen << '\n';
        edellinen = nykyinen;
    }
}


/*
while (cin>>current) toteutuu niin monta kertaa, kun operaattori cin>>current
on voimassa eli niin kauan kuin on olemassa luettavia merkkejä käyttäjän
syötteessä. String-muuttujalle operaattori >> lukee tyhjällä välillä
(whitespace) eroteltuja sanoja. Loopin voi katkaista painamalla Ctrl-C
(katkaisee koko ohjelman toiminnan) tai Ctrl-Z (Unix-koneissa Ctrl-D).
*/

/*
kuvaus ohjelman toiminnasta lauseelle "ankka ui ui vedessä":

  Näytölle tulostuu ohjeteksti.

  edellinen sana on tyhjä, ohjelma lukee sanaa ankka
  while-looppi on voimassa. Ohjelma vertaa sanaa ankka tyhjään ja if-lause
   jää suorittamatta, sillä ankka ei ole sama kuin tyhjä tila
  string-muuttujan edellinen arvoksi sijoitetaan nykyinen sana eli ankka.

```

```
43    edellinen on ankka, ohjelma lukee käyttäjän syöttämän sanan ui.
44    while-looppi, ohjelma vertaa sanoja ankka ja ui, if-lause ei toteudu ja
45     muuttujan edellinen arvoksi syötetään merkkijono ui.
46
47    edellinen on ui, ohjelma lukee syötteen ui
48    while-looppi, ohjelma vertaa sanoja ui ja ui ja if-lause toteutuu, jolloin
49    näytölle tulostuu teksti: Toistettu sana: ui (rivinvaihto)
50    muuttujan edellinen arvoksi sijoitettaan sana ui
51
52    edellinen on ui, ohjelma lukee sanan vedessä ja vertaa ui ja vedessä
53     keskenään, if-lause ei toteudu, sijoitetaan edellinen muuttujan arvoksi
54     vedessä
55
56    while-looppi ei toteudu, koska käyttäjän syötteessä ei ole enempää sanoja.
57  */
```

Esimerkissä käytettiin `if`-lausetta, joka toteutuu silloin, kun sen argumenttina eli `()`-sulkujen sisällä oleva ehto on tosi. Ehtorakenteista puhumme lisää seuraavassa luvussa.

Ohjelmoinnissa on tyypillistä, että jo olemassa olevaa koodia parannellaan ja muokataan pitkänkin ajan kuluessa. Seuraavassa on hieman paranneltu versio nimiharjoituksesta, joka kertoo myös sen, kuinka mones sana oli kirjoitettu kaksi kertaa.

2.5.2 Esimerkki: nimiharjoitus 2

nimiharjoitus2.cpp

```
1  /* ============================================================================ */
2  /*                                                                              */
3  /*   nimiharjoitus2.cpp                                                         */
4  /*   (c) 2011 Riku Järvinen                                                     */
5  /*                                                                              */
6  /*   Description                                                                */
7  /*   Tarkastaa kirjoitetusta lauseesta, ovatko peräkkäiset sanat samoja.
8      kertoo lisäksi, kuinka mones sana oli kirjoitettu kaksi kertaa.*/
9  /* ============================================================================ */
10
11 #include "../std_lib_facilities.h"
12
13 int main()
14 {
15     int sanojen_lkm = 0; // alustetaan sanoista kirjaa pitävä laskuri
16     cout << "Kirjoita teksti\x84, niin ohjelma etsii kahdentuneet sanat.\n";
17     string edellinen = " ";  //edellinen sana: alkuarvo ei ole sana
18     string nykyinen;
19     while (cin>>nykyinen) {
20         ++sanojen_lkm; //aina loopin alussa lisätääm counteria yhdellä
21         if(edellinen == nykyinen) // onko edellinen saman kuin nykyinen
22             cout << "Sana numero " << sanojen_lkm
```

```
23                  << " toistettu: " << nykyinen << '\n';
24          edellinen = nykyinen;
25      }
26  }
```

2.6 Yhdistelmäoperaattorit

Sijoittaminen ja lisääminen/vähentäminen sekä sijoittaminen ja jakaminen/kertominen voidaan yhdistää yhdeksi operaattoriksi, esimerkiksi

```
a += 8;    // tarkoittaa a = a+8;
b -= 7     //             b = b-7;
c /= 2     //             c = c/2;
d *= 2     //             d = 2*d;
```

Binäärioperaatioille eli operaatioille, jotka ovat kahden muuttujan välillä, voidaan aina kirjoittaa em. tavalla.

2.7 Tyypit ja niiden varaama tila muistissa

Kertausta ja hieman uutta:

- *Objekti* on osa tietokoneen muistia, jonne varastoidaan määrätyn tyypin arvo.
- *Muuttuja* on nimetty objekti eli nimetty paikka koneen muistissa.
- *Tyyppi* määrittää muuttujan mahdollisten arvojen joukon sekä ne operaatiot, joita muuttujalle voidaan tehdä
- *Arvo* on muistissa sijaitseva bittijono, joka tulkitaan tyypin perusteella.
- **deklaraatio** (*declaration*) on lauseke, joka antaa nimen objektille eli määrittää muuttujan.

Muistamme, että tyyppejä ovat mm. `bool`, `char`, `int`, `double` ja `string`. Tyypistä riippuen tietokone varaa kullekin tietyn määrän muistia, yleensä taulukon 2 mukaisesti. Taulukon 2 bittimäärät tarkoittavat binääriarvoja, joten esimerkiksi

TAULUKKO 2: Tyypit ja niiden viemä tila muistissa

Tyyppi	**tavut** (bytes)	**bitit** (bits)	**tavut graafisesti**
bool	1	8	X
char	1	8	X
int	4	32	XXXX
double	8	64	XXXXXXXX

`int`-tyypin muuttujaa voisi esittää seuraavanlainen bittijono:

10110010 00101110 00001100 00010101

Bittien sisältämän tiedon eli binääridatan tulkinta riippuu täysin tyypistä. Esim. data 10010010 voidaan tulkita joko kirjaimena (`char`) tai kokonaislukuna (`int`).

`string`-tyypin muuttujaan tallentuu tieto merkkijonon merkkien määrästä sekä itse kirjaimista. Merkkijonon varaama bittimäärä riippuu näin ollen sen pituudesta.

Taulukossa 2 `bool` vie kahdeksan bittiä, vaikka käytännössä se voi sisältää vain kaksi mahdollista arvoa (true tai false). Tämä johtuu tietokonemuistin teknisistä ominaisuuksista: **pienin toimiva tiedon tallennusyksikkö on tavu**.

2.8 Tyyppimuunnokset

Objektin tyyppi voidaan muuntaa toiseen tyyppiin kahdella tavalla, joko **turvallisesti** tai **ei-turvallisesti** (*safe and unsafe type conversions*). Turvallisessa muunnoksessa tietoa ei häviä, ei-turvallisessa sitä voi hävitä. Tällöin ohjelma voi toimia ei-toivotulla tavalla. Toisaalta, jotkut ohjelmat on suunniteltu toimimaan normaalisti ei-turvallisten muunnosten kautta (ei käsitellä tällä kurssilla). Kun yleisesti puhutaan **tyyppien turvallisuudesta** (*type safety*), tarkoitetaan, että kirjoitetussa ohjelmassa muuttujien tyypit on asetettu oikein niin, että tietoa ei vahingossa katoa (ts. ei käytetä ei-turvallisia tyyppimuunnoksia).

Pääsääntöisesti muunnos on turvallinen, jos tietoa siirretään pienemmästä tiedon varaamiseen tarkoitetusta yksiköstä suurempaan. Esimerkkejä turvallisista muunnoksista ovat mm.

```
bool → char
bool → int
bool → double
char → int
char → double
int  → double
```

Mikäli `int` on erittäin suuri, niin muunnos `int` → `double` saattaa aiheuttaa pientä epätarkkuutta tietokoneen laskentatehosta riippuen.

Ei-turvallisia muunnoksia ovat mm.

```
double → int
double → char
double → bool
int    → char
int    → bool
char   → bool
```

Ongelmana `double` → `int` -muunnoksessa on kokorajoituksen lisäksi pyöristys (aina alaspäin) ja `double` → `char` -muunnoksessa suuri double ei mahdu chariin, jolloin saadaan epämääräinen arvo. Vastaavasti myös `int` → `char` jne. Esimerkiksi PC:ssä char-arvot ovat välillä $[-128, 127]$ eli 256 eri arvoa (tulee suoraan, kun laskee $2^8 = 256$.

Ei-turvalliset muunnokset ovat olemassa osittain historiallisista syistä (on olemassa koodia, jossa niitä on käytetty aiemmin, esim. C-kielellä) eli jotkut ohjelmat tarvitsevat niitä toimiakseen oikein. Jos omissa ohjelmissa epäilet sitä, että muunnoksessa häviää tietoa, niin kannattaa tarkastaa muuttujan arvo ajamalla ohjelma ennen kuin arvon sijoittaa eteenpäin johonkin lausekkeeseen. Katsotaan esimerkkinä muutamia tyyppimuunnoksia. Ohjelmaa on kommentoitu melko rajusti, mutta kannattaa lueskella kaikki kohdat tarkasti niin huomaa mielenkiintoisia asioita.

2.8.1 Esimerkki: tyyppimuunnokset

tyyppikonversiot.cpp

```
/* ========================================================================== */
/*
      tyyppikonversiot.cpp                                                  */
/*   (c) 20.2.2011 Riku Järvinen */
/*                                                                           */
/*   Description
     Testaillaan tyyppimuunnoksia: käyttäjän syöttämä liukuluku muunnetaan
     kokonaisluvuksi ja merkiksi ja vertaillaan arvoja. */
/* ========================================================================== */

#include "../std_lib_facilities.h"

int main()
{
    double a = 0;
    cout << "\nSeuraavat selitykset p\x84tev\x84t seuravaaville kirjaimille:\n"
         << "Alkuper\x84inen double on a\n"
         << "double k\x84\x84nnettyn\x84 int-arvoksi on b\n"
         << "int k\x84\x84nnettyn\x84 char-merkiksi on c\n"
         << "char-merkin int-arvo on d.\n"
         << "k\x84\x84nnetty int-arvo doubleksi on e\n"
         << "\nKirjoita ruudulle haluttu luku tai luvut!\n";
    while (cin>>a) {
        int b = a;
        char c = b;
        int d = c;
        double e = d;
        cout << "\na==" << a
             << "\nb==" << b
             << "\nc==" << c
             << "\nd==" << d
             << "\ne==" << e << "\n";
    }
}
```

Seuraavassa hieman testituloksia edellisestä ohjelmasta.

```
/* Kokeillaan erilaisia inputteja ja katsotaan, miten numerot muuttuvat.

a==2
b==2
c==?
d==2

a==3
```

```
b==3
c==?
d==3

a==-2
b==-2
c==¦
d==-2
e==-2

a==-15
b==-15
c== ±
d==-15
e==-15

 ok - ohjelma näyttää ainakin alustavasti toimivan kunnolla

a==127
b==127
c==¦
d==127

a==128
b==128
c==Ç
d==-128

 havaitaan charin yläraja eli 127, seuraava arvo 128 menee alarajalle -128

a==-128
b==-128
c==Ç
d==-128

a==-129
b==-129
c==¦
d==127

 havaitaan, että charin alaraja on -128, sillä -129 menee ylärajalle 127

a==1
b==1
c==?
d==1

a==257
b==257
c==?
```

```
59 d==1
60
61 a==513
62 b==513
63 c==?
64 d==1
65
66  havaitaan syklisyys arvoissa, eli sama char tulee aina 256:n välein ja näin
67  ollen myös sama d eli tässä tapauksessa ykkönen. Luku 256 tulee bittimäärästä:
68  char sisältää 8 bittiä eli 2^8=256 erilaista arvoa (merkkiä).
69
70 a==59.9
71 b==59
72 c==;
73 d==59
74 e==59
75
76  havaitaan doublen pyöristyminen kokonaisluvuksi ja tiedon häviäminen
77  muunnoksen kautta
78
79 a==0
80 b==0
81 c==
82 d==0
83 e==0
84
85 a==-2.14748e+009
86 b==-2147483648
87 c==
88 d==0
89 e==0
90
91 a==2.14748e+009
92 b==-2147483648
93 c==
94 d==0
95 e==0
96
97  huomataan int-muuttujan lukualue, eli (2^32) = 4294967296, tarkoittaa
98  käytännössä väliä [-2147483648, +2147483648].
99 */
```

2.9 Kysymyksiä luvusta 2

1. Mitä tarkoittaa termi *prompt*?
2. Mitä operaattoria käytetään, kun arvo luetaan muuttujaan?
3. Mitä tarkoittaa `\n` ja mihin sitä käytetään?

4. Mikä lopettaa tekstin syötön `string`-tyypin muuttujaan?
5. Mikä lopettaa arvon syötön `int`-tyypin muuttujaan?
6. Mikä on *objekti*?
7. Mikä on *literaali*?
8. Mikä on *muuttuja*?
9. Mitkä ovat tyypilliset koot muuttujille `char`, `int` ja `double`?
10. Mitä mittayksikköä käytetään muistipaikoissa muuttujan varaamalle muistille?
11. Mitä tarkoittaa `string`-muuttujien yhdistäminen (+) ja kuinka se toimii C++:ssa?
12. Millaisia nimiä C++:ssa voi käyttää? millaisia sallituja nimiä ei kannata käyttää?
13. Mitä tarkoittaa *tyyppien turvallisuus* (type safety) ja miksi se on tärkeä ominaisuus?
14. Mitä ongelmia voi aiheuttaa muunnos `double` → `int`?
15. Määrittele sääntö, joka perustelee sen, että tyyppimuunnos on turvallinen.

2.10 Tehtäviä luvusta 2

1. Kirjoita ohjelma, joka ei varsinaisesti tee mitään, vaan ainoastaan määrittelee muutamia muuttujia, osa virheellisesti ja osa oikein (esimerkiksi `int double =0;` tai vastaavaa). Katso, kuinka kääntäjä reagoi.
2. Kirjoita ohjelma, joka pyytää käyttäjää syöttämään kaksi kokonaislukua. Ohjelman tulee määrittää luvuista pienempi ja suurempi sekä laskea summa, erotus, tulo ja osamäärä sekä kertoa ne käyttäjälle.
3. Muunna edellisen tehtävän ohjelmaa siten, että se kysyy käyttäjältä liukuluvun (`double`) kokonaisluvun sijasta. Vertaa uuden ja vanhan ohjelman tulostamia arvoja. Eroavatko ne toisistaan?
4. Kirjoita ohjelma, joka kysyy käyttäjältä kolme kokonaislukua ja tulostaa ne pilkuilla erotettuina suuruusjärjestyksessä pienimmästä suurimpaan. Jos esimerkiksi käyttäjä kirjoittaa 3 5 1 niin ruudulle tulostuu 1, 3, 5. Jos kaksi arvoa ovat yhtä suuria, niin niiden keskinäisellä järjestyksellä ei ole merkitystä.
5. Tee samantyyppinen ohjelma kuin edellisessa tehtävässä, mutta korvaa numerot `string`-merkkijonoilla.
6. Kirjoita ohjelma, joka muuntaa kirjaimin kirjoitetut numerot (esimerkiksi ”nolla”) numeroarvoiksi kuten 0. Kun käyttäjä kirjoittaa numeron kirjaimin, niin ohjelman tulee tulostaa vastaava numeroarvo. Toteuta tämä numeroille 0, 1, 2, 3 ja 4 ja laita lisäksi ohjelma tulostamaan teksti ”tuntematon numero” jos käyttäjä syöttää jotain muuta kuin edellämainitut.

3 Laskentaa

Luvussa keskitytään mm. ohjelmoinnin yleisiin tavoitteisiin, lausekkeisiin (valinta ja iteraatio), funktioihin sekä vektoreihin.

3.1 Laskennan perusteet

Laskenta tarkoittaa jonkin **ulostulon** (*output*) tuottamista jostakin **sisääntulosta** (*input*). Sisääntulo voi olla esim. näppäimistöltä, hiireltä, syntetisaattorilta, mittausanturilta tai vastaavalta. Ohjelmoinnissa kiinnostavimmat näkökohdat ovat **I/O** (*input/output*) toisesta ohjelmasta tai toisesta ohjelman osasta. Perusidea hyvässä ohjelmointisuunnittelussa on määrittää se, *millä tavalla eri osat toimivat keskenään ja millaisia rooleja niillä on datan tuottamisessa ja käyttämisessä*.

Ohjelman osaan, esimerkiksi funktioon, menevää sisääntuloa kutsutaan **argumentiksi** (*argument*; näitä voi olla useita) ja ulostuloa **tulokseksi** tai **palautusarvoksi** (*return value*).

3.2 Ohjelmoinnin yleiset tavoitteet

Laskentaprosessin kolme tärkeintä peruselementtiä ovat laskujen **oikeellisuus**, **yksinkertaisuus** ja **tehokkuus** tässä järjestyksessä. Ohjelmia korjataan ja kehitetään jatkuvasti, joten suunnittelu kannattaa tehdä jo alusta lähtien kunnolla, ennakoiden mahdollisia tulevaisuuden laajennuksia niin hyvin kuin vain voi. Hyvä suunnittelu tarkoittaa

- selkeää rakennetta koodille
- suurien, globaalien muutosten tekemistä tarvittaessa nopeasti ja edullisesti
- verraten pientä virheiden määrää. Tämä säästää myös virheiden korjaamiseen kuluvaa aikaa.

Kaiken kaikkiaan ohjelman vankka, hyvä rakenne säästää turhaa työtä myöhemmin.

Kuinka tällainen perusta voidaan saada aikaiseksi? Ajatuksena on jakaa suuri ohjelma pieniin osiin

1. **abstrahoimalla** (*abstraction*): käytetään valmiita kirjastoja, funktioita ja algoritmeja eli ei määritellä itse kaikki mahdollisia tarvittavia työkaluja.
2. **erottelemalla toisistaan ohjelman loogiset osat ja käsittelemällä ne omina aliohjelminaan**, jotka yhdessä muodostavat kokonaisen, toimivan ohjelman.

Miksi suunnittelua tehdään ja kuinka välttämätöntä se on? Kun koodia alkaa olemaan reippaat 1 000 000 riviä, niin ohjelman kehittäminen ei onnistu ilman loogista järjestystä. Jos koodarit vaihtuvat, kuten usein tapahtuu, niin uusien henkilöiden on saatava selvää ohjelman rakenteista ja toiminnoista, yleensä nopealla aikataululla.

3.3 Ilmaisut

Ilmaisu (*expression*) laskee arvon **operandien** ja **operaattoreiden** perusteella ja on ohjelmakoodin perusblokki. Esimerkiksi muuttujan määrittely muotoa

```
int luku = 3
```

on ilmaus, samoin

```
double c = a + b
```

mikäli `a` ja `b` on määritelty (ja alustettu) aiemmin. Ilmauksesta saadaan **lauseke** (*statement*) laittamalla sen perään puolipiste, jolloin tässä yhteydessä puhutaan **ilmaisulausekkeesta** (*expression statement*).

Ilmauksessa `int luku = 3` *luku*-sana on objektin (muistipaikan) nimi, sen tyyppi on `int` ja siitä käytetään joskus termiä **vasen arvo** (*left value, lval*). Vastaavasti numero 3 on muuttujan `luku` arvo eli **oikea arvo** (*right value, rval*). Ilmaisuissa ovat matematiikan tavalliset säännöt ovat voimassa, kuten mm. +, -, *, / ja niin edelleen. Tämän lisäksi on suositeltavaa käyttää tarpeen mukaan yhdistelmäoperaattoreita `+=`, `-=`, `/=` ja `*=` yksinkertaistamaan toimintoja. Arvon lisäämisessä yhdellä kannattaa käyttää ilmausta `++arvo;`.

Vakioiden ilmaiseminen C++:ssa tehdään `const`-merkinnällä, esimerkiksi

```
const double pi = 3.14159;
```

määrittää vakion π arvoksi 3.14159. Kannattaa määrittää vakio alussa ja käyttää sen nimeä pelkän numeroarvon sijaan, sillä se selventää koodia. Lisäksi vakiota on helpompi muuttaa tarvittaessa (määrittely vain yhdessä paikassa). Pääsääntöisesti vain lukuja 0 ja 1 saa käyttää koodin seassa itsestäänselvissä ja varmasti oikein tulkittavissa jutuissa (esimerkiki palautusarvo, joka kertoo, suorittiko ohjelma toimintonsa normaalisti loppuun saakka).

Operaattoreita on paljon ja niitä oppii käytön kautta. Huomaa, että ilmaus `a<b<c` ei tarkoita mitään eikä sitä tule käyttää, sillä `a<b` palauttaa `bool`-tyypin arvon, joka on `true` tai `false` ja sen vertaaminen arvoon `c` ei ole mielekästä.

Tyyppimuunnoksissa kannattaa kiinnittää huomiota erityisesti **jakolaskuun**: esimerkiksi `5/2` antaa tulokseksi `2`, jos muuttujien tyyppeinä on `int`, ja `5.0/2` antaa `2.5`. Tarvittaessa kääntäjä korottaa `int`-tyypin muuttujan arvon laskuoperaation ajaksi `double`-tyypin arvoksi ja `char`-tyypin arvon `int`-tyypin arvoksi, jotta laskussa ei menetetä informaatiota. Käytännössä tämä huomataan vaikkapa Celcius-Fahrenheit-konversiossa

```
double celcius = 0;
double fahrenheit = 0;
cin >> celcius;
fahrenheit = 9.0/5*celcius + 32;
```

jossa on syytä havaita luku `9.0` yläkerrassa (`double`-tyyppiä).

3.4 Lausekkeet, ehtolauseet ja toistorakenteet

Olemme jo tutustuneet ilmaisulausekkeeseen eli ilmaisuun, jonka perään on laitettu puolipiste. Katsotaan lisäksi hieman **valinta- ja ehtolauseita** (`if` ja `switch`) sekä **toistorakenteita** (`while` ja `for`). Näiden avulla saadaan runsaasti lisää toiminnallisuutta ohjelmiin.

3.4.1 Ehtolauseet

If-lausetta käytetään, kun halutaan valita useasta eri vaihtoehdosta. Lause on rakenteeltaan

```
if(ilmaus) lauseke else lauseke
```

Katsotaan esimerkkinä ohjelma, joka muuntaa valuuttoja toisiksi valuutoiksi.

3.4.2 Esimerkki: Valuutan muuntaminen

valuuttamuunnin.cpp

```
/* ========================================================================== */
/*                                                                            */
/*   valuuttamuunnin.cpp                    */
/*   (c) 2011 Riku*/
/*                                                                            */
/*   Description                                                              */
/*   muuntaan dollareita, puntia ja ruotsin kruunuja euroiksi */
/* ========================================================================== */

#include "../std_lib_facilities.h"

int main()
{
  // määritellään valuuttakurssit, vakioita
  const double dollar_euro = 0.721;
  const double pound_euro  = 1.177;
  const double krona_euro  = 0.114;
  double rahamaara = 0;
  char rahayksikko = ' ';
  // kysytään käyttäjältä rahamäärä ja yksikkö
  cout << "\n[Valuuttamuunnin]\n\n"
       << "Ohjelma muuntaa rahasumman euroiksi. Kirjoita ruudulle"
       << " raham\x84\x84r\x84 ja\nsen j\x84lkeen yksikk\x94"
       << " eli jokin seuraavista:\n"
       << "\n d == USA:n dollari\n"
       << " k == Ruotsin kruunu\n"
       << " p == Englannin punta\n"
       << "\n>> ";
  cin  >> rahamaara >> rahayksikko;
  // valintalause ja muunnokset
  if (rahayksikko == 'd')
    cout << "\n" << rahamaara << " USA:n dollaria on yhteens\x84 "
         << dollar_euro*rahamaara
         << " euroa.\n";
  else if (rahayksikko == 'k')
    cout << "\n" << rahamaara << " Ruotsin kruunua on yhteens\x84 "
         << krona_euro*rahamaara
         << " euroa.\n";
  else if (rahayksikko == 'p')
    cout << "\n" << rahamaara << " Englannin puntaa on yhteens\x84 "
         << pound_euro*rahamaara
         << " euroa.\n";
  else
    cout << "\nKirjoitit yksik\x94n, jota en tunnista";
}
```

Ohjelma muuntaa USA:n dollareita, Englannin puntia sekä Ruotsin kruunuja Euroihin. Koodin alussa määritetään vakioina muunnoskertoimet dollareista euroihin (`dollar_euro`), punnista euroihin sekä kruunuista euroihin. Lisäksi määri-

tellään muunnettava rahamäärä sekä kirjaimet, jotka kuvaavat valuuttoja. Kirjain merkitään yksittäisten lainausmerkkien ' ' sisään (alussa tyhjä arvo).

Tulostetaan kehote ja luetaan käyttäjältä muunnettava määrä sekä yksikkö. Jaetaan mahdolliset tapauksen erilaisiin `if`-lauseisiin käyttäjän syöttämän yksikön mukaan. Huomaa, että yhtäsuuruusmerkkinä on kaksi tavallista yhtäsuuruusmerkkiä ja käyttäjän syöttämä kirjain tulee yksittäisten lainausmerkkien sisään. Käytetään `else`-rakenteen lauseen paikalla uutta `if`-lausetta, jolloin muodostuu sisäkkäisten if-lauseiden rakenne: ohjelma tarkistaa, onko ensimmäinen voimassa, jos ei niin sitten siirtyy aina seuraavaan. Jos käyttäjä syötti jonkin muun yksikön kuin jonkin annetuista, niin viimeinen `else`-lause tulostaa ruudulle ilmoituksen siitä, että muunto ei tällä kerralla onnistunut.

> **Huomio.** Jos `if`-rakenteen "sisällä" on useita eri lausekkeita, tulee ne laittaa samaan **blokkiin** seuraavasti:
>
> ```
> if (ehto) {
> lause1;
> lause2;
> lause3;
> ...
> ...
> }
> ```
>
> Jos blokki unohtuu, niin ehto koskee vain ensimmäistä lauseketta ja loput lausekkeet suoritetaan tarkastamatta, onko ehto voimassa. Sama blokkisääntö pätee myös muille vastaavantyyppisille rakenteille (mm. `while`- ja `for`-lauseet).

`if`-lauseen sijaan voidaan käyttää joissakin tapauksissa **switch-lausetta**, jossa muuttujan mahdolliset arvot käsitellään selkeästi erillisinä tapauksina. Joskus `switch` voi auttaa selventäämään koodia paremmin kuin `if`-lause, sillä se on rakenteeltaan seuraavanlainen:

```
switch (muuttuja)
case case_label_1:
  lauseke;
  ...
  lauseke;
  break;
case case_label_2:
  lauseke;
  ...;
  break;
  ...
case case_label_n:
  lauseke;
  ...;
  break;
default;  // jos ei ollut mikään case niin sitten tämä vaihtoehto
  lauseke;
  ...;
  break;
```

`Switch`-lauseen jokainen `case` pitää aina lopettaa lausekkeeseen `break;`, jotta lausee suoritus päättyy ja palataan takaisin pääohjelmaan. Katsotaan äsken esitetty valuuttaamuunnin `switch`-muodossa, jotta lauseen käyttö ilmenee paremmin. Lisätään ohjelman käyttäjälle mahdollisuus muuntaa myös Japanin jenejä euroiksi.

3.4.3 Esimerkki: valuutan muuntaminen, osa 2

valuuttamuunnin2.cpp

```
1  /* ========================================================================== */
2  /*                                                                            */
3  /*   valuuttamuunnin2.cpp                                                     */
4  /*   (c) 2011 Riku                                                            */
5  /*                                                                            */
6  /*   Description                                                              */
7  /*   muuntaa jenejä, dollareita, puntia ja ruotsin kruunuja euroiksi,
8       switch-lause */
9  /* ========================================================================== */
10
11 #include "../std_lib_facilities.h"
12
13 int main()
14 {
15   // määritellään valuuttakurssit, vakioita
16   const double dollar_euro = 0.721;
17   const double pound_euro = 1.177;
18   const double krona_euro = 0.114;
19   const double yen_euro = 0.009;
20   double rahamaara = 0;
21   char rahayksikko = ' ';
22   // kysytään käyttäjältä rahamäärä ja yksikkö
23   cout << "\n[Valuuttamuunnin]\n\n"
24        << "Ohjelma muuntaa rahasumman euroiksi. Kirjoita ruudulle"
25        << " raham\x84\x84r\x84 ja\nsen j\x84lkeen yksikk\x94"
26        << " eli jokin seuraavista:\n"
27        << "\n d == USA:n dollari\n"
28        << " k == Ruotsin kruunu\n"
29        << " p == Englannin punta\n"
30        << " y == Japanin jeni\n"
31        << "\n >> ";
32   cin  >> rahamaara >> rahayksikko;
33   // switch-lause ja muunnokset
34   switch (rahayksikko) {
35   case 'd':
36     cout << "\n" << rahamaara << " USA:n dollaria on yhteens\x84 "
37          << dollar_euro*rahamaara << " euroa.\n";
38     break;
39   case 'k':
40     cout << "\n" << rahamaara << " Ruotsin kruunua on yhteens\x84 "
41          << krona_euro*rahamaara << " euroa.\n";
42     break;
43   case 'p':
44     cout << "\n" << rahamaara << " Englannin puntaa on yhteens\x84 "
45          << pound_euro*rahamaara << " euroa.\n";
46     break;
```

```
47    case 'y':
48      cout << "\n" << rahamaara << " Japanin jeni\x84 on yhteens\x84 "
49           << yen_euro*rahamaara << " euroa.\n";
50      break;
51    default:
52      cout << "\nKirjoitit yksik\x94n, jota en tunnista";
53      break;
54    }
55  }
```

Ohjelman looginen rakenne on hyvin pitkälti identtinen `if`-lauseella tehdyn ohjelman kanssa. Muutama yksityiskohta tulee kuitenkin huomioida.

- `Switch`-lauseen muuttujan arvon tulee (tässä vaiheessa oppimista) olla joko tyyppiä `int` tai `char`, ts. se ei voi olla esimerkiksi `string`-tyyppiä.
- `Case` labelien tulee olla vakioita, muuttujia ei saa käyttää. Yhdelle tapaukselle voidaan käyttää useampia `case` labeleita (vaikkapa parilliset luvut jostakin lukujoukosta).
- Jokainen `case` päättyy `break;`-lausekkeeseen.

3.4.4 Toistorakenteet

Kun jokin asia toistetaan useita kertoja pienillä muutoksilla, voidaan käyttää valmiita ohjelmointikielen rakenteita sen sijaan, että kirjoitettaisiin hirveä määrä samanlaista koodia koodin perään. **Toistorakenteita** (*repetition*) ovat **while**- ja **for-lauseet**, ja toistoa varten tarvitaan

- *looppi* (*loop*), mitä toistetaan
- *looppimuuttuja* (*loop/control variable*), joka pitää kirjaa siitä, montako kertaa looppi on käyty läpi
- looppimuuttujan *alkuarvo* (*initialization value*)
- loopin *lopettamisehto* (*termination criterion*), esimerkiksi looppimuuttujan jokin maksimiarvo.

`while`-lausetta toistetaan niin kauan, kun looppimuuttujaan liittyvä alkuehto on voimassa. Tässä kannattaa olla tarkkana, sillä monta kertaa kone on saatu jumiin ”ikuisella `while`-loopilla”. Komentoriviohjelman toiminnan saa Window-sissa lopetettua painamalla Ctrl-C, jos näin käy. Tarkastellaan yksinkertaista `while`-looppia, jossa tulostetaan ruudulle kirjaimia ja niitä vastaavia numeroarvoja.

3.4.5 Esimerkki: aakkosten tulostus ruudulle, osa 1

kirjaintaulukko.cpp

```
1  /* ============================================================================ */
2  /*                                                                              */
3  /*   kirjaintaulukko.cpp                                        */
4  /*   (c) 2011 Riku Järvinen */
5  /*                                                                              */
```

```
6  /*    Description                                                          */
7  /*    tulostaa kirjaimia ja niitä vastaavia indeksejä taulukkoon   */
8  /* ========================================================================= */
9
10 #include "../std_lib_facilities.h"
11
12 int main()
13 {
14   int i = 0;    // looppimuuttujan alustus
15   while (i<26) {        // looppiehto ja blokin aloitus
16     cout << char('a'+i) << '\t' << int('a'+i) << "\n";    // char('a'+1)=b jne.
17     ++i;                // looppimuuttujan kasvattaminen yhdellä
18   }
19 }
```

Huomaa, kuinka ohjelmassa otetaan `char`-arvo eli merkki annetun indeksin mukaan eli `char('a'+i)` ja vastaavalla tavalla `int`-arvo `int('a'+1)`. Käydään läpi yhteensä 26 aakkosta, aloittaen kirjaimesta `a` ja tulostetaan kirjain ja indeksi ruudulle. Rakenteen lopussa kasvatetaan looppimuuttujaa `i` yhdellä kirjoittamalla `++i;`.

`while`-lauseen ohella voidaan käyttää `for`-lausetta, jossa `while`-rakenteesta poiketen kontrollimuuttujan (looppimuuttujan) hallinta on sijoitettu heti `for`-sanan jälkeen. `for`-lause on aina yhtäpitävä jonkin `while`-lauseen kanssa, eli on olemassa `while`-lause siten, että täysin sama asia voidaan sanoa myös sen kautta. `for`-lauseen käyttö on järkevää lähinnä yksinkertaisissa ja selkeissä rakenteissa. Kirjoitetaan edellinen kirjaintaulukkoesimerkki nyt käyttämällä `for`-lausetta.

3.4.6 Esimerkki: aakkosten tulostus ruudulle, osa 2

kirjaintaulukko2.cpp

```
1  /* ========================================================================= */
2  /*                                                                           */
3  /*    kirjaintaulukko2.cpp                                        */
4  /*    (c) 2011 Riku Järvinen */
5  /*                                                                           */
6  /*    Description                                                            */
7  /*    tulostaa kirjaimia ja niitä vastaavia indeksejä taulukkoon, for-lause   */
8  /*    lisäksi isot kirjaimet ja numerot 1-9. */
9  /* ========================================================================= */
10
11 #include "../std_lib_facilities.h"
12
13 int main()
14 {
15   for (int i=0; i<26; ++i)        // looppiehto ja blokin aloitus
16     cout << char('a'+i) << '\t' << int('a'+i) << "\n";
17   // laitetaan yksi rivi väliä ennen isoja kirjaimia
18   cout << "\n";
19   for (int i=0; i<26; ++i)        // looppiehto ja blokin aloitus
20     cout << char('A'+i) << '\t' << int('A'+i) << "\n";    // char('A'+1)=b jne
```

```
21    cout << "\n";
22    for (int i=0; i<9; ++i)
23      cout << char('1'+i) << '\t' << int('1'+i) << "\n";
24  }
```

`for`-lauseen toistojen määrä ja rakenne ilmoitetaan argumenttina tavallisten sulkujen sisällä. Looppimuuttujan alustus, loopin lopetuskriteeri sekä looppimuuttujan kasvattaminen erotetaan toisistaan puolipisteillä tavalliseen tapaan. For-lauseen rakenne toistetaan ainoastaan ensimmäiselle lausekkeelle, joka rakennetta seuraa, tai koko seuraavalle blokille, jos sellainen on olemassa (rajattu kaarisuluilla).

Testaa ohjelman toimintaa käytännössä. Huomaat, että kirjaimien ja numeroiden viereen tulostuvat luvut ovat **indeksiarvoja:** jokaista `char`-tyypin muuttujaa eli merkkiä vastaa jokin numeerinen indeksi, jonka perusteella merkki voidaan tunnistaa yksikäsitteisesti, kun käytettävä merkistö on tunnettu. Koska `char` on kooltaan vain yhden tavun, niin erilaisia merkkejä voidaan sen avulla esittää yhteensä 256 kappaletta. Tästä syystä monet erikoismerkit, kuten mm. skandinaaviset kirjaimet, aiheuttavat ongelmia joissakin sovelluksissa.

3.5 Funktiot

Funktio on *nimetty ja järjestetty joukko lausekkeita*, jotka yhdessä toteuttavat jonkin loogisen operaation. Funktio voi esimerkiksi laskea annettujen lukujen perusteella arvon ja palauttaa tuloksen (korkeintaan yhden ja tietyn tyyppisen) pääohjelmalle. Esimerkiksi funktio

```
int toinenpotenssifunktio(int x) {return x*x;}
```

laskee annetun kokonaisluvun `x` neliön. Tässä tapauksessa funktion käyttö eli **kutsuminen** tapahtuu kirjoittamalla

```
toinenpotenssifunktio(luku)
```

sopivaan osaan koodia. Kutsu toimii kuten sen paikalle olisi kirjoitettu vastaava kokonaisluku. Voidaan vaikkapa kirjoittaa summauslauseke

```
int a = 0;
int b = 3;
int a += toinenpotenssifunktio(b);
cout << a << "\n";
```

Edellinen tulostaa ruudulle arvon 9. Yleisesti ottaen funktion kielioppi eli **syntaksi** voidaan kirjoittaa seuraavasti:

```
tyyppi nimi (parametrilista) {funktiorunko}
```

Parametrilistaan tulevat ne argumentit, jotka funktiolle pitää antaa, jotta se voi tehdä tehtävänsä. Kaikki tarvittavat argumentit on annettava ja lisäksi niiden tulee olla oikeaa tyyppiä, muuten kääntäjä ilmoittaa virheestä. Parametrilista voi olla myös tyhjä.

Funktio, joka ei palauta mitään, on tyyppiä `void`. Esimerkiksi

```
void kirjoita_joitain() {cout << "Jotain\n";}
```

kirjoittaa ruudulle sanan `Jotain` ja vaihtaa riviä, kun funktiota kutsutaan.

Miksi funktioita käytetään? Tähän on olemassa useita hyviä syitä, jotka tulevat ilmi vasta sitten, kun ohjelmoinnista on enemmän kokemusta. Listataan näistä muutamia.

- Usein toistuvat operaatiot kirjoitetaan lyhyemmin, jolloin koodi on selvempää ja luettavampaa. Tämä säästää aikaa ja vaivaa, jos ja kun virheitä ilmenee.
- Funktion kuvaileva nimi tekee koodista ymmärrettävämpää. Esimerkiksi standardikirjaston funktio `sqrt()` laskee luvun neliöjuuren ja se on aika paljon helpompi huomata tekstin seasta kuin iso kasa koodia, jossa selitetään itse laskualgoritmia.
- Funktioiden käyttö parantaa tekstin rakennetta ja helpottaa sen hahmottamista, kun erityyppiset loogiset operaatiot on määritelty toisistaan selvästi erilleen.

Hyvä funktio tekee loogisesti yhden asian eli vaikkapa joko laskee tai tulostaa tai jotain muuta. Se ei tee kaikkia näitä asioita, vain yhden. Monimutkainen ja epälooginen funktio aiheuttaa vain sekaannusta, vaatii oman dokumentaationsa eik selvennä koodin rakennetta (tämä oli funktion käytön idea alun perin...).

Funktiorungon tarkastelu ei useinkaan ole tarpeen, jos tiedämme, minkä toimenpiteen funktio suorittaa. Jotta funktioita voidaan käyttää koodissa, ei niiden täydellisiä määritelmiä tarvitse aina kirjoittaa uudestaan koodin sekaan näkyviin, vaan voidaan käyttää **funktiodeklaraatioita** (*declaration*), jotka ovat muotoa

```
int square(int);
```

Puolipiste päättää deklaraation. Kun funktio otetaan käyttöön, voi koodin alkuun yksinkertaisesti kirjoittaa

```
#include int square(int);
```

Varsinainen funktion toiminnallisuus voi olla määritelty kirjoitettavan koodin ulkopuolella (esimerkiksi standardikirjaston funktiot). Huomaa tässä deklaraation ja *määritelmän* (*definition*) ero: ensimmäinen ei sisällä yksityiskohtaista tietoa funktion rungosta. Deklaraation etu laajoissa ohjelmissa on se, että suuri osa epäoleellisesta koodista voidaan pitää poissa näkyvistä keskityttäessä johonkin oleelliseen .

3.6 Vektorit

Monissa mielekkäissä ohjelmissa tietoa kysytään käyttäjältä ja tallennetaan myöhempää käyttöä varten. Tieto tallennetaan objekteihin koneen muistiin, ja yksi tällaisista tiedon varastoista on nimeltään **vektori** (*vector*). Vektoriin syötetään lukuja ja se voidaan ajatella ikään kuin taulukon yhtenä rivinä, jossa luvut ovat eri sarakkeissa. Esimerkiksi kokonaislukuvektori voidaan kirjoittaa näin:

```
vector<int> vektori(3);
vektori[0] = 2;
vektori[1] = 6;
vektori[2] = -29;
```

Vektorissa `vektori` on kolme elementtiä ja ne ovat `int`-tyyppiä. Indeksointi alkaa nollasta. Huomaa, että tyyppi merkitään kulmasuluilla <> ja vektorin elementin numero hakasuluilla `[]`. Kun vektorille syötetään tietoa, pitää aina käyttää oikeaa tyyppiä ja viitata jo olemassa olevaan vektorin indeksiin (muuten tulee ”mystinen” `Range Error`, joka viittaa siihen, että mentiin vahingossa vektorin määrittelyalueen ulkopuolelle). Vektorin voi alustaa myös kirjoittamalla

```
vector<string> merkkijonovektori(2,tyhja);
```

Edellinen laittaa jokaisen elementin alkuarvoksi merkkijonon ”tyhja” ja elementtejä on vektorissa yhteensä kaksi. Vektoriin tallentuu tieto vektorin koosta (kuinka monta elementtiä se sisältää), elementtien tyypistä sekä elementtien sisällöstä.

Vektoria voi kasvattaa eli sen loppuun voi lisätä tarvittaessa uusia elementtejä. Tämä on tyypillistä, sillä usein ei etukäteen tiedetä, kuinka monta dataelementtiä yhteensä tulee olemaan, vaikkapa fysikaalisessa mittauksessa. Kirjoitetaan seuraavaa:

```
vector<double> numerodataa;   // ei määritä kokoa, vain tyypin!
numerodataa.push_back(3.4);   // lisää elementin, arvo 3.4, indeksi [0]
numerodataa.push_back(-8.3); // lisää elementin, arvo -8.3, indeksi [1]
```

Funktio `push_back()` sijoittaa elementin vektorin loppuun. Sitä sanotaan vektorin **jäsenfunktioksi** (*member function*) ja sen kutsuminen tapahtuu pistenotaatiolla eli `numerodataa.push_back(arvo);`. Yleisesti jäsenfunktioiden käytön syntaksi on muotoa

```
objektin_nimi.jäsenfunktion_nimi(argumentit)
```

Muita vektorin jäsenfunktioita ovat mm. seuraavat:

- `size()`, joka kertoo vektorin koon eli elementtien lukumäärän
- `begin()`, joka viittaa vektorin ensimmäiseen elementtiin
- `end()`, joka viittaa vektorin viimeiseen elementtiin
- `at()`, joka kertoo vektorin elementin sijainnin

Täydellinen lista jäsenfunktioista löytyy helposti netistä, kun etsii tietoa C++:n vektoreista.

`begin()`- ja `end()`-funktioiden käyttö on näppärää esimerkiksi algoritmin `sort()` kanssa, jolla voidaan järjestää vektorin elementit:

```
vector<int> vektori(2);
vektori[0] = 4;
vektori[1] = -2;
sort(vektori.begin(),vektori.end()); // pienimmästä suurimpaan
```

`sort()`-algoritmi on osa standardikirjastoa. Se järjestää luvut suuruusjärjestykseen pienimmästä suurimpaan ja esimerkiksi merkkijonot aakkosjärjestykseen.

Katsotaan vektorin käyttöä parin esimerkin kautta. Ensimmäisessä pyydetään käyttäjää syöttämään lämpötiloja ja lasketaan niiden mediaani ja jälkimmäisessä sovelletaan `sort()`-algoritmia sensuroimaan epämukavia sanoja.

3.6.1 Esimerkki: Lämpötilojen mediaanin laskeminen

lampotilamediaani.cpp

```
1 /* ============================================================================ */
2 /*                                                                              */
3 /*   lampotilamediaani.cpp                                                      */
4 /*   (c) 2011 Riku                                                            */
5 /*                                                                              */
6 /*   Description                                                                */
7 /*    Laskee kirjoitettujen lämpötilojen mediaanin matem. määr. mukaisesti */
8 /* ============================================================================ */
9
```

```
10 #include "../std_lib_facilities.h"
11
12 int main()
13 {
14     vector<double> temps;     // lämpötilavektori
15     double temp;
16     cout << "Kirjoita l\x84mp\x94tiloja, muu kuin luku lopettaa. \n";
17     while (cin>>temp)        // looppi päättyy kun syötetään muuta kuin luku
18         temps.push_back(temp);    // kirjoitetut lämpötilat talteen vektoriin
19     sort(temps.begin(),temps.end());    // lämpötilat suuruusjärjestykseen
20     if (temps.size()%2 == 0)      // jos lämpötiloja on parillinen määrä
21         cout << "Mediaani on "
22              << 0.5*(temps[temps.size()/2-1]+temps[temps.size()/2]) << endl;
23     if (temps.size()%2 == 1) // jos lämpötiloja on pariton määrä
24         cout << "Mediaani on "
25              << temps[temps.size()/2] << endl;     // pyöristää alaspäin, ok.
26 }
```

Koodin alussa määritellään lämpötilavektori `temps`, johon käyttäjä lisää lämpötiloja `while`-loopin kautta. Kun käyttäjä kirjoittaa lämpötilan `temp`, niin jäsenfunktio `push_back(temp)` lisää sen vektorin `temps` viimeiseksi elementiksi. `while`-looppi katkeaa, kun käyttäjä ei enää syötä lukuja vaan jotain muuta.

Lämpötilat järjestää pienimmästä suurimpaan `sort`. Mediaania varten tarkastelemme `if`-lauseen avulla kaksi eri tapausta:

- Jos annettuja lukuja on parillinen määrä, niin mediaani on järjestetyn joukon kahden keskimmäisen luvun keskiarvo.
- Jos lukuja on pariton määrä, niin mediaani on järjestetyn joukon keskimmäinen luku.

Parillisuus testataan ottamalla vektorin koon ja luvun 2 jakojäännös. Mikäli se on nolla, niin vektorissa on parillinen määrä elementtejä; vastaavasti arvo 1 kuvaa paritonta määrää. Huomaa, että `if`-lauseen jälkeen ei tarvita blokkimerkintää (kaarisulkuja), sillä ehdon jälkeen tulostetaan vain yksi (vaikkakin kahden rivin mittainen) `cout`-lauseke.

3.6.2 Esimerkki: Epämukavien sanojen sensurointi

epamukavatsanat.cpp

```
1  /* ========================================================================== */
2  /*                                                                            */
3  /*   epamukavatsanat.cpp                                          */
4  /*   (c) 2011 Riku Järvinen */
5  /*                                                                            */
6  /*   Description                                                              */
7  /*   tulostaa VIRHE, kun käyttäjä syöttää sanan, joka on ns. mustalla listalla
8          */
9  /* ========================================================================== */
10
11 #include "../std_lib_facilities.h"
12
```

```
13 int main()
14 {
15   /* sanalista epämukavista sanoista, määritellään vektorina */
16   vector<string> epamukaviasanoja(5);
17   epamukaviasanoja[0] = "ankka";
18   epamukaviasanoja[1] = "kala";
19   epamukaviasanoja[2] = "auto";
20   epamukaviasanoja[3] = "lentokone";
21   epamukaviasanoja[4] = "simulaattori";
22   vector<string> sanoja;   // määrittelee vektorin nimeltä sanoja
23   string sana;
24   cout << "Kirjoita ruudulle sanoja tyhj\x84ll\x84 v\x84lill\x84 erotettuina."
25        << " Kun olet kirjoittanut kaikki haluamasi sanat, paina Enter, Ctrl-Z"
26        << " ja Enter.\n";
27   while (cin>>sana)        // looppi lisää syötetyt sanat vektoriin
28     sanoja.push_back(sana);
29   sort(sanoja.begin(),sanoja.end()); // järjestää syötetyt sanan aakkosjärj.
30   cout << "\n";
31   for (int i=0; i<sanoja.size(); ++i) // käy läpi sanavektorin
32   /* kahden if-lauseen rakenne, joka tarkastaa ensin sanojen samanlaisuuden ja
33      vielä erikseen sen, onko sana mikaan vektorin epamukaviasanoja sanoista */
34     if (i==0 || sanoja[i-1]!=sanoja[i])  // poistaa toistetut sanat
35       if (sanoja[i] == epamukaviasanoja.at(0) ||
36           sanoja[i] == epamukaviasanoja.at(1) ||
37           sanoja[i] == epamukaviasanoja.at(2) ||
38           sanoja[i] == epamukaviasanoja.at(3) ||
39           sanoja[i] == epamukaviasanoja.at(4))
40         cout << "VIRHE" << "\n";
41       else
42         cout << sanoja[i] << "\n";
43 }
```

Alussa tehdään sanalista epämukavista sanoista, joka määritellään vektorina `epamukaviasanoja`. Lisäksi määritellään toinen vektori `sanoja`, johon tallennetaan käyttäjän syöttämät sanat. `sort` järjestää käyttäjän syöttämät sanat aakkosjärjestykseen, jonka jälkeen sanat käydään läpi `for`-lauseella.

Ensimmäinen `if`-lause pitää huolen siitä, että tulostetaan vain kerran useaan otteeseen esiintynyt sana. Toisessa `if`-lauseessa määritellään tulostukseksi sana `VIRHE`, mikäli läpikäytävä sana on jokin alussa annetuista epämukavista sanoista. Muussa tapauksessa (`else`-rakenne) tulostetaan annettu sana ruudulle ja laitetaan perään rivinvaihto.

3.7 Esimerkkejä lauserakenteista ja vektoreista

Tämä osio sisältää muutamia harjoituksia Stroustrupin kirjan pohjalta. Alkuperäisiin tehtävänantoihin nähden esimerkkejä voi olla jonkin verran muunneltu ja täydennetty. Koodeja ei ole (ainakaan vielä) kommentoitu koodi-ikkunan ulkopuolella.

Kaikki koodit toimivat Windows-ympäristöissä MinGW:n 32-bittiselle tehdyllä kääntäjällä.

3.7.1 Esimerkki: Matkan pituuden laskeminen

matkanpituus.cpp

```
/* ========================================================================== */
/*                                                                            */
/*   ex4_3.cpp*/
/*   (c) 2011 Riku */
/*                                                                            */
/*   Description                                                              */
/*   määrittää etappien välisiä etäisyyksiä ja koko matkan pituuden. */
/* ========================================================================== */

#include "../std_lib_facilities.h"

int main()
{
    vector<double> etaisyydet;
    double etaisyys = 0;
    cout << "Kerro matkan etappien et\x84isyydet (km), niin lasken matkan"
         << " pituuden.\nJokin muu kuin numero lopettaa.\n";
    while (cin >> etaisyys)
        etaisyydet.push_back(etaisyys);  // luetaan etäisyydet vektoriin
    double summa = 0; // määritellään summa, johon etäisyydet laitetaan
    for (int i=0;i<etaisyydet.size();++i)  // käydään vektori läpi
        summa += etaisyydet[i]; // lisätään etäisyys summaan
        cout << "Koko matkan pituus on " << summa << " km.\n";
    double pienin = 10000;  // älyttömän iso luku, joka varmasti muuttuu
    double suurin = 0;      // älyttömän pieni luku, joka varmasti muuttuu
    for (int j=0;j<etaisyydet.size();++j) // etsitään ääriarvot
    {   // blokki, jotta kaikki lauseet tulevat huomioiduiksi
        if (etaisyydet[j] < pienin)
            pienin = etaisyydet[j];
        if (etaisyydet[j] > suurin)
            suurin = etaisyydet[j];
    }
    cout << "Pisimm\x84n etapin pituus on " << suurin << " km\n";
    cout << "Lyhyimm\x84n etapin pituus on " << pienin << " km\n";
}
```

3.7.2 Esimerkki: Yksinkertainen laskinohjelma

laskinohjelma.cpp

```
/* ========================================================================== */
/*                                                                            */
/*   laskinohjelma.cpp                                                        */
/*   (c) 4.5.2011 Riku Järvinen*/
/*                                                                            */
```

```
6  /*    Description                                                          */
7  /*    Laskinohjelma, joka laskee peruslaskutoimitukset kahdelle luvulle*/
8  /* ========================================================================= */
9
10 #include "../std_lib_facilities.h"
11
12 int main()
13 {
14     // otsikko ohjelmalle
15     cout << "\n\n[LASKINOHJELMA]\n\n";
16     // kehote käyttäjälle
17     cout << "Kirjoita ruudulle kaksi desimaalilukua ja jokin merkeist\x84 "
18          << "\n+, -, *, / tai %. Ohjelma laskee annettujen lukujen v\x84lisen"
19          << "\nlaskutoimituksen.\n\n"
20          << ">>";
21     // määritellään luvut
22     double a, b;
23     char merkki;
24     // luetaan luvut
25     cin >> a >> b >> merkki;
26     cout << "\n";
27     // tarkastellaan eri tapaukset
28     switch(merkki) {
29     case '+':
30         cout << "Lukujen " << a << " ja " << b << " summa on " << a+b << ".\n";
31         break;
32     case '-':
33         cout << "Lukujen " << a << " ja " << b << " erotus on " << a-b << ".\n";
34         break;
35     case '*':
36         cout << "Lukujen " << a << " ja " << b << " tulo on " << a*b << ".\n";
37         break;
38     case '/':
39         cout << "Lukujen " << a << " ja " << b << " osam\x84\x84r\x84 on "
40              << a/b << ".\n";
41         break;
42     case '%':
43         // muutetaan double int-tyyppiin moduloa varten
44         int a_toint = a;
45         int b_toint = b;
46         cout << "Lukujen " << a_toint << " ja " << b_toint
47              << " jakoj\x84\x84nn\x94s on " << a_toint%b_toint << ".\n";
48         break;
49     }
50 }
```

3.7.3 Esimerkki: lukujen arvaaja

arvaaja.cpp

```
/* ========================================================================== */
/*                                                                            */
/*   arvaaja.cpp*/
/*   (c) 2011 Riku */
/*                                                                            */
/*   Description                                                              */
/*   arvaa numeron väliltä 1-100 kysymällä kysymyksiä 7 kappaletta */
/* ========================================================================== */

#include "../std_lib_facilities.h"

int main()
{
    double numero = 50;
    char merkki = 'a';
    cout << "Ajattele jotakin lukua v\x84lilt\x84 1-100. Selvit\x84n, mit\x84"
         << " numeroa ajattelet. \n";
    cout << "Onko ajattelemasi numero \n\nsuurempi kuin 50, \nyht\x84"
          " suuri kuin 50, vai \npienempi kuin 50?\n\n"
         << "Kirjoita\n\n +  jos on suurempi tai yht\x84suuri"
         << "\n -  jos on pienempi\n =  jos ajattelemasi numero on 50.\n\n";
    cin >> merkki;
    while (true) // tarvitaan, että break voi lopettaa loopin kun luku löytyy
    {
        if (merkki == '=')
            {
            cout << "Ajattelit numeroa 50\n";
            break;
            }
        if (merkki == '+')
            numero = numero + 50.0/2; // muistetaan jakolaskussa desimaalipilkku
        if (merkki == '-')
            numero = numero - 50.0/2;
        cout << "Onko ajattelemasi numero suurempi kuin "
             << numero << "?\n";
        cout << "Kirjoita\n\n +  jos on suurempi"
             << "\n -  jos ei ole\n";
        cin >> merkki;
        if (merkki == '+')
            numero = numero+50.0/4;
        if (merkki == '-')
            numero = numero-50.0/4;
        cout << "Onko ajattelemasi numero suurempi kuin "
             << numero << "?";
        cout << "\nKirjoita\n\n +  jos on suurempi"
             << "\n -  jos ei ole\n";
        cin >> merkki;
        if (merkki == '+')
            numero = numero+50.0/8;
```

```
50          if (merkki == '-')
51              numero = numero-50.0/8;
52          cout << "Onko ajattelemasi numero suurempi kuin "
53               << numero << "?";
54          cout << "\nKirjoita\n\n +  jos on suurempi"
55               << "\n -  jos ei ole\n";
56          cin >> merkki;
57          if (merkki == '+')
58              numero = numero+50.0/16;
59          if (merkki == '-')
60              numero = numero-50.0/16;
61          cout << "Onko ajattelemasi numero suurempi kuin "
62               << numero << "?";
63          cout << "\nKirjoita\n\n +  jos on suurempi"
64               << "\n -  jos ei ole\n";
65          cin >> merkki;
66          if (merkki == '+')
67              numero = numero+50.0/32;
68          if (merkki == '-')
69              numero = numero-50.0/32;
70          cout << "Onko ajattelemasi numero suurempi kuin "
71               << numero << "?";
72          cout << "\nKirjoita\n\n +  jos on suurempi"
73               << "\n -  jos ei ole\n";
74          cin >> merkki;
75          if (merkki == '+')
76              numero = numero+50.0/64;
77          if (merkki == '-')
78              numero = numero-50.0/64;
79          cout << "Onko ajattelemasi numero suurempi kuin "
80               << numero << "?";
81          cout << "\nKirjoita\n\n +  jos on suurempi"
82               << "\n -  jos ei ole\n";
83          cin >> merkki;
84          if (merkki == '+')
85              numero = numero+50.0/128;
86          if (merkki == '-')
87              numero = numero-50.0/128;
88          // määritetään lopullinen numero kokonaislukuna
89          int kokonaislukuna_numero = numero; // määritellään numero ↩
                kokonaislukuna
90          int lopullinen_numero = 0;   // alustetaan lopullinen tulostettava ↩
                numero
91          if (numero-kokonaislukuna_numero >= 0.5)  //  pyöristys ylöspäin
92              lopullinen_numero = kokonaislukuna_numero+1;
93          if (numero-kokonaislukuna_numero < 0.5)   // pyöristys alaspäin
94              lopullinen_numero = kokonaislukuna_numero;
95          cout << " Ajattelemasi numero on " << lopullinen_numero;
96          break; // lopettaa while (true) -loopin
97      }
```

```
98 }
```

3.8 Kysymyksiä luvusta 3

1. Mikä on laskutoimitus (computation)?
2. Mitä tarkoittavat sisääntulo (input) ja ulostulo (output) laskutoimitukselle? Anna esimerkki.
3. Mitkä kolme asiaa ohjelmoijan on syytä pitää mielessä laskutoimituksia tehdessään?
4. Mitä ”expression” eli ilmaus tekee?
5. Mikä ero on ilmauksella (expression) ja lausekkeella (statement) monisteen perusteella?
6. Mikä on vakioilmaus (constant expression)?
7. Mikä on literaali (literal)?
8. Mitä operaattoreita voidaan käyttää kokonaisluvuille ja liukuluvuille?
9. Mitä operaattoreita voidaan käyttää kokonaisluvuille, mutta ei liukuluvuille?
10. Mitä operaatioita (joitakin niistä kaikista) voidaan käyttää string-tyypin muuttujille?
11. Milloin ohjelmoijan on syytä käyttää switch-lausetta if-lauseen sijaan?
12. Mitkä ovat tavallisimpia ongelmia/rajoituksia switch-lauseen käytön kanssa?
13. Mitä for-loopin argumentin eri osat tekevät ja missä järjestyksessä ne suoritetaan?
14. Milloin tulee käyttää for-looppia ja milloin while-looppia?
15. Kuinka tulostetaan merkin (char) numeerinen arvo?
16. Mitä tarkoittaa rivi char foo(int x) funktion määrittelyssä?
17. Milloin kannattaa määritellä erillinen funktio jollekin ohjelman osalle?
18. Mikä on vektorin kolmannen elementin indeksi?
19. Kuinka kirjoitetaan for-looppi, joka tulostaa jokaisen vektorin elementin?
20. Mitä tekee vector<char> alphabet(26); ?
21. Kuvaile, mitä push_back() tekee vektorille.
22. Mitä tekevät vektorin jäsenfunktiot begin(), end() ja size()?
23. Kuinka voit järjestää vektorin elementit?

3.9 Tehtäviä luvusta 3

1. Kirjoita opintukilaskuri, joka kertoo, kuinka paljon tietyn ikäinen henkilö saa valtiolta opintotukea. Ohjelman tulee ottaa syötteinä seuraavat asiat:

   - tuen saajan ikä
   - asuuko tuen saaja kotona vai omassa asunnossaan
   - jos oma asunto asunnon vuokra

   Opintotuki koostuu opintorahasta ja asumislisästä. Valtion takaama opintolaina ei suoraan kuulu opintotukeen, eikä sitä tarvitse tässä huomioida. Ajan tasalla olevat tiedot opintotuen ja asumislisän suuruudesta saat KELAn nettisivuilta.

   Yksinkertaisuuden vuoksi voidaan olettaa, että tuen saajalla ei ole lapsia, hänen asuntonsa vuokra on ”kohtuullinen” (saa tukena 80% vuokrasta) ja hänellä ei ole merkittäviä tuloja, jotka vaikuttaisivat opintotuen suuruuteen.

2. Toteuta luvun 3.7 lukuarvaaja-esimerkki for-lausetta käyttäen. (Vinkki: for-loopin sisällä tarvitset funktiota `pow2,i`, joka kuvaa kakkosen i:ttä potenssia, kun looppimuuttuja on i).

3. Tavallisella shakkilaudalla on 64 ruutua. Asetetaan riisinjyviä laudalle siten, että ensimmäiseen ruutuun tulee yksi jyvä, toiseen kaksi, kolmanteen neljä ja niin edelleen. Kirjoita ohjelma, joka kysyy käyttäjältä täytettävien ruutujen lukumäärän ja tulostaa riisinjyvien yhteismäärän näissä ruuduissa. Selkeyden vuoksi kannattaa kirjoittaa for-looppi, jota pyöritetään käyttäjän syöttämään ruutumäärään asti ja tulostetaan ruudulle jokaisen käyntikerran jälkeen jyvien lukumäärä.

4. Mikäli käytit jyvien lukumäärälle int-tyyppiä, niin edellisessä ohjelmassa huomasit kenties sen, että 32. ruudun kohdalla ja sen jälkeen määrät eivät enää pidä paikkaansa, koska lukualue loppuu kesken. Kokeile muuttaa int→double ja testaa toimivuutta. Lisää ohjelma tulostamaan ruudulle ilmoitus siitä ruudusta, jossa jyvien kokonaismäärä ylittää 1000., 1000000. ja 1000000000. jyvän rajat.

5. Tee ohjelma, jolla voit pelata konetta vastaan Kivi, paperi, sakset -peliä. Yksinkertaisuuden vuoksi voit antaa ohjelmalle arvot, joilla se vastaa käyttäjälle (aina samat arvot, esimerkiksi vektorissa). Jos haluat, niin voit myös syöttää koneelle arvot, joita se käyttää vastatessaan, mahdollisesti eri järjestyksessä kuin ne on syötetty. Ohjelman tulee ilmoittaa käyttäjälle pelin tulos jokaisen yrityksen jälkeen (voitto, tappio, häviö) lyhyellä tulosteella.

4 Bjarne Stroustrupin ajatuksia liittyen ohjelmointiin

Ohjelma pitää olla suunniteltu siten, että käyttäjän syöttämä virheellinen input ei aiheuta merkittäviä ongelmia (antaa virheilmoituksen tms.). Virheitä tosin käsitellään paremmin vasta seuraavassa luvussa.

ohjelmoinnin hyvyyttä ei mittaa mahdollisimman monimutkaisen koodin kirjoittaminen, vaan kyky kirjoittaa vaaditun työn tekevä ja mahdollisimman yksinkertainen koodi.

Tietokone on laite, jota voidaan ohjelmoida tekemään teoriassa millaisia laskutoimituksia tahansa (toimintaa ei ole ennalta määrätty!). Kun tämä liitetään yhteen reaalimaailman laitteiden kanssa, saadaan laitteet tekemään teoriassa mitä tahansa.