ATK I
OHJELMOINNIN PERUSTEET
76314P

LUENTOMATERIAALI

Mauri Puoskari ja Mikko Saarela
Oulun yliopisto
Fysikaaliset tieteet/ Teoreettinen fysiikka
Linnanmaa
90570 Oulu

syksy 2000

Sisältö

Kuvat

Luku 1

Ohjelmointi

1.1 Ohjelmointikielet

Ohjelmointikielet voidaan luokitella joko käyttötavan mukaan (1) tulkkaaviin ja (2) käännettäviin kieliin tai niiden käyttötarkoituksen mukaan.

1.1.1 Ohjelmointikielten luokitteluja

Tulkkaavat kielet

- Tulkkiohjelma lukee käyttäjän antamia ohjelmointikäskyjä ja suorittaa ne.
- Käskyjen kokeilu ja ohjelman testaus on helppoa.
- Ohjelman suorittamiseksi tarvitaan itse ohjelman lisäksi tulkki.
- Ohjelmien suoritus on hitaampaa kuin käännettävissä kielissä.

Käännettävät kielet

- Kääntäjäohjelma (*compiler*) kääntää käyttäjän tekemän ohjelman *konekielelle*.
- Käskyjen ja ohjelmien kokeilu vaatii aina uudelleenkäännöksen (ja linkityksen).
- Tuotettu konekielinen ohjelma (`Windowsissa .EXE`) on itsenäisesti suoritettava komento.
- Ohjelman suoritus on nopea (verrattuna vastaavaan tulkkaavaan ohjelmaan).

Yleiskielet

- Voidaan käyttää periaatteessa kaikenlaisten ohjelmien tekoon: numeeriset tehtävät, merkkijonojen käsittely, grafiikan teko jne.

Erikoiskielet

- Erikoistuneet jonkin tietyn ongelma-alueen käsittelyyn, esim. tietokantoihin, ladontaan, jne.

Korkean tason kielet

- Asiat pyritään ilmaisemaan käyttäjälle tutuilla tavoilla.
- Ohjelmat eivät (yleensä) riipu käytettävästä koneesta eivätkä käyttöjärjestelmästä.

Alhaisen tason kielet

- Ohjelmien rakenne noudattaa koneen (prosesorin) ominaisuuksia.
- Ohjelmat koneesta riippuvia.
- Voidaan tehdä sellaista, mihin korkean tason kielten ilmaisut eivät riitä.
- Tehokkaita (nopeita ja kompakteja) ohjelmia.
- Ei yleensä kannata käyttää (muuten kuin pakkotilanteessa).

Proseduraaliset kielet

- Ohjelmoija antaa ohjeet, *miten* haluttu asia toteutetaan.

Deklaratiiviset kielet

- Ohjelmoija ilmoittaa, *mitä* haluaa.

Olioihin perustuvat kielet

- Ohjelmoija luo olioita, jotka käsittelevät tutkittavaa dataa halutulla tavalla. Oliot sisältävät sekä datan, että käsittelymenetelmän samassa paketissa.

Strukturaaliset kielet

- Ohjelmarakenteet (*if, repeat,* proseduurit,...) jäsentävät ohjelman.

Modulaariset kielet

- Ohjelmat voidaan jakaa osiin, *moduleihin*.
- Moduleita voidaan koota aliohjelma*kirjastoiksi*.

1.1.2 Ohjelmointikieliä

C ja C++

- Yleiskieli.
- Käännettävä kieli.
- Strukturaalinen ja modulaarinen kieli.
- Paljon valmiita ohjelmakirjastoja.
- Paljon alhaisen tason operaatioita (esimerkiksi bittien käsittelyoperaatioita).
- Ammattilaisen kieli.
- C++ soveltuu olio-ohjelmointiin.

Java

- Yleiskieli.
- Käännettävä kieli, käännetään koneesta riipumattomalle kielelle
- Olioihin perustuva kieli
- Strukturaalinen ja modulaarinen kieli
- Käytetään erityisesti internet-sovellutusten ohjelmointiin

FORTRAN

- Yleiskieli, erikoisesti numeerisiin tehtäviin soveltuva.
- Käännettävä kieli.
- Paljon valmiita numeriikkakirjastoja.

BASIC

- Yleiskieli.
- Tulkkaava kieli.
- Paljon valmiita käskyjä esimerkiksi grafiikkaa ja ääntä varten.
- Ei suosi strukturaalista ohjelmointia.
- Ei ole modulaarinen.
- Mikrotietokoneharrastajien suosiossa.

PASCAL

- Yleiskieli.
- Käännettävä kieli.
- Strukturaalinen kieli.
- Pakottaa “oikeaoppiseen” ohjelmointiin.
- Käytettään erityisesti ohjelmoinnin opetukseen.

COBOL

- Kaupallishallinnollisiin sovelluksiin erikoistunut kieli.
- Käännettävä kieli.

ASSEMBLER

- Alhaisen tason kieli.
- Symbolinen konekieli: yhdestä *assembly*kielisestä käskystä syntyy yksi konekielinen käsky.

SQL

- Tietokantojen käsittelyyn tarkoitettu kieli.
- Pääasiallisesti deklaratiivinen kieli.

TEX

- Ladontakieli.
- Erikoisen hyvä teknisen tekstin ladonnassa.

FORTH

- Yleiskäyttöinen kieli.
- Tulkkaava kieli.
- Laajennettava kieli: ohjelmoija määrittelee uusia sanoja.

LISP

- Yleiskieli, mutta erikoisen sopiva tekoälysovelluksiin.
- Tulkkaava/käännettävä kieli.
- Ohjelmoija kirjoittaa *funktioita*.

PROLOG

- Erikoistunut logiikka- ja tekoälysovelluksiin.
- Tulkkaava/käännettävä kieli.
- Ohjelmoija kirjoittaa *tietokannan* ja *päättelysäännöt*.

Luku 2

C- kieli

2.1 Alkeet

Tietokoneen käsittelee tietoa, joka esitetään binäärilukuina. Binäärilukujärjestelmä on kaksitila- systeemi, jonka tilat ovat 1 ja 0 (esimerkiksi virta kulkee tai ei kulje, piste on magnetoitu tai ei ole ja jne). Tätä pienintä tiedon yksikköä kutsutaan bitiksi, joita ryhmitellään suuremmiksi kokonaisuuksiksi:

- **Bitti** (b,bit) Datan pienin yksikkö (0 tai 1).
- **Tavu** (B,byte) Datan perusyksikkö: 8 bittiä.
- **Sana** (word) CPUn rakenteen kannalta luontevin datan määrä; mikroissa tavallisesti 32 bittiä
- **Kilotavu** (kB tai KB) 1 kB on 2^{10} tavua eli $1kB = 1024B$
- **Megatavu** (MB) 1 MB on 2^{10} kilotavua eli $1MB = 1024kB$
- **Gigatavu** (MB) 1 MB on 2^{10} Megatavua eli $1GB = 1024MB$

Numeroiden, kirjaimien ja erikoismerkkien esittämiseksi binäärimuodossa on tehty sopimus siitä, mitkä bittijonot vastaavat kutakin merkkiä. Sovittu ns. ASCII- koodi (American Standard Code for Information Interchange) käyttää 7 bittiä esittämään kutakin merkkiä. Koska ASCII- koodista sovittiin Amerikassa, siinä ei ole otettu huomioon muiden kuin englannin kielen tarvitsemien aakkosten koodaus. Jos käytetään vain seitsemää bittiä, ne voidaan asettaa $2^7 = 128$ erilaiseen järjestykseen. Kunkin eri järjestykseen asetetun bittijonon voidaan sopia vastaavan yhtä systeemin tuntemasta merkistä.

Muiden kielien kanssa käytettävissä järjestelmissä ASCII- koodi on laajennettu 8- bittiseksi, mutta valitettavasti eri laitevalmistajat käyttävät erilaisia koodeja muunmuassa skandinaavisten aakkosten esittämiseen. Kahdeksan bitin avulla voidaan esittää 2^8 eli 256 erilaista merkkiä; siten skandinaavisten merkkien ASCII- koodit kymmenjärjestelmän lukuina esitettyinä ovat suurempia kuin luku 128. ASCII-koodi on esitetty liitteessä A.

Yksi merkki vie yhden tavun verran tilaa muistista. Yksi konekirjoitustusarkki tekstiä tarvitsee puolestaan muistitilaa noin 2KB, jolloin yhteen megatavuun mahtuu noin 500 sivua tekstiä.

2.1.1 Ohjelman rakenne

C- kielessä ohjelma koostuu yhdestä tai useammasta osasta, pääohjelmasta ja aliohjelmista. C- kielessä kaikki ohjelman osat (myös pääohjelma) ovat funktioita.

C- ohjelma voidaan jakaa useisiin tiedostoihin, joista kukin voi sisältää yhden tai useampia funktioita. Kukin tiedosto koostuu siten globaaleista määrittelyistä (eli määrittelyistä, jotka ovat yhteisiä useille funktioille) ja funktioista. Funktioita **on oltava** aina **vähintään yksi**: pääohjelma `main`. Aliohjelmat samoin kuin globaalit määrittelyt voivat puuttua kokonaan tai niitä voi olla mielivaltainen määrä.

Suositeltavaa on jakaa C- ohjelma eri tiedostoihin siten, että asiallisesti yhteen kuuluvat funktiot ja määrittelyt tulevat aina yhteen tiedostoon. Lisäksi on syytä välttää hyvin pitkiä tiedostoja.

2.1.2 Ensimmäinen ohjelma

Ensimmäinen tehtävä aina uutta ohjelmointikieltä opeteltaessa on tehdä lyhyt ohjelma, jolla tulostetaan jotakin näytölle. Tavallinen esimerkki on seuraava ohjelma, joka tulostaa viestin "Hello World"ja rivinvaihdon näyttöön:

```
/* Esimerkkiohjelma Hello World */
#include <stdio.h>
main()
{
   printf("Hello World\n");
}
```

Tässä pienessä ohjelmassa esiintyy jo monia tyypillisiä C- kielen rakenteita:

- Globaalit määrittelyt tulevat tiedoston alkuun; `include`- direktiivillä otetaan mukaan kirjasto `stdio` ("standard input and output). Tämä kirjasto sisältää tulostusfunktion `printf` määrittelyn.
- Pääohjelman nimen tulee olla `main`.
- Funktion runko kirjoitetaan aaltosulkujen sisään (eli se on tyypiltään *lohko* ja se koostuu määrittelyjen ja lauseiden jonosta, joista kukin on kirjoitettu omalle rivilleen.
- Funktio `printf` huolehtii tulostuksesta.
- Funktioiden argumentit kirjoitetaan sulkujen sisään. Jos funktiolla ei ole ollenkaan argumentteja,niin sulut kuitenkin tarvitaan, jotta kääntäjä osaa tunnistaa funktion rakenteen oikein.
- Funktion kutsu muodostuu funktion nimestä ja suluissa olevasta argumenttilistasta; tässä tapauksessa funktion nimi on `printf` ja argumenttilistana on merkkijono, joka on kirjoitettava lainausmerkkeihin, kuten kaikki muutkin merkkijonot.
- Merkintä `\n` tarkoittaa rivinvaihtoa, ja sitä käsitellään C- kielessä yhtenä merkkinä.
- Lauseet lopetetaan puolipisteellä lukuunottamatta direktiiveja, kuten `#include` ja `#define` y.m.

Ohjelman suoritus alkaa pääohjelmasta (toisin sanoen funktiosta jonka nimi on `main`) ja päättyy, kun pääohjelman suoritus päättyy.

C- kielen funktioiden yleinen muoto on:

```
globaalit määrittelyt
funktion_nimi(argumenttilista määrittelyineen)
{
   funktion sisäiset määrittelyt
   funktion_runko
}
```

2.1.3 Muuttujat ja sijoituslausekkeet

Muuttuja kuvaa muistipaikan sisältöä. Muuttujien määrittelyissä kerrotaan millaista tietoa muuttujat voivat sisältää: kokonaislukuja (`int`), reaalilukuja (`float`), merkkejä (`char`), ja jne.

Tarkastellaan esimerkkiä

```
/* Esimerkkiohjelma Luennon pituus */
#include <stdio.h>
main()
{
   int tunnit, minuutit, sekunnit;

   tunnit = 2;
   minuutit = 60 * tunnit;
   sekunnit = 60 * minuutit;
   printf("Luennon pituus on:\n");
   printf("  %d tuntia \n", tunnit);
   printf("  %d minuuttia\n", minuutit);
   printf("  %d sekuntia\n", sekunnit);
}
```

Tässä esimerkissä on määritelty kolme kokonaislukumuuttujaa: `tunnit, minuutit, sekunnit`.

Sijoituslauseissa,

```
tunnit = 2;
minuutit = 60 * tunnit;
sekunnit = 60 * minuutit;
```

muistipaikkoihin tunnit, minuutit ja sekunnit sijoitetaan arvoja, eli muuttujat saavat arvonsa.

Sijoituslauseen yleinen muoto:

```
muuttuja = sijoitettava lauseke;
```

Sijoitettava lauseke voi olla yksittäinen luku, algebrallinen lauseke, kuten edellä, tai yhdistelmä muuttujista, operaattoreista, funktioista jne.

Esimerkissä suoritetaan yksikkömuunnokset minuuteille ja tunneille ja tulokset kirjoitetaan näyttöön **printf** funktiota käyttäen.

Tulostuslauseen yleinen muoto:

```
printf(muotoilu, arg1, arg2, ..., argn );
```

missä muotoiluosa on lainausmerkkien väliin kirjoitettu merkkijono, joka määrää tulostuksen ulkoasun. Argumentteja pilkuilla erotettuna voi olla mielivaltainen määrä tai ei yhtään. Muotoilua `%d` käytetään kokonaislukujen tulostukseen.

Esimerkki, joka sisältää merkki- ja reaalilukumuuttujia

```
/* Tulostetaan
   1) merkin ASCII koodi
   2) reaalilukujen summa
*/

#include <stdio.h>
main()
{
        /* määrittelyt */
```

```
  float a, b;
  float summa;
  char c;
        /* sijoituslauseet */
  c = 'A';
  a = 1.0;
  b = 2.0;
  summa = a + b;
        /* tulostus */
  printf("Merkin  %c", c);
  printf("ASCII- koodi on %d\n", c);
  printf("\n");  /* ylimaarainen rivinvaihto */
  printf("Lukujen %f ja  %f summa on %f\n", a, b, summa);
}
```

Tulostuksen muotoilussa on huomattava, että merkkimuuttujaa tulostettaessa `%c` muotoilulla tulostuu merkki ja `%d` muotoilulla puolestaan kyseistä merkkiä vastaava ASCII- koodi, joka on pieni kokonaisluku (pienempi kuin 256). Muotoilukoodilla `%f` tulostetaan desimaalilukuja.

2.1.4 Perustietotyypit

C- kielen perustietotyypit ovat:

`char`	merkkitieto
`int`	kokonaisluku
`float`	reaaliluku
`double`	kaksoistarkkuuden reaaliluku

Kokonaisluvut

Kokonaisluvun `int` esitykseen käytetään yksi sana eli yleensä 32 bittiä. sen arvoalue on $-2^{31}, \ldots, 2^{31}-1$ eli suurin sallittu kokonaisluku on 2147483647 ja pienin on -2147483648. Jos halutaan käyttää suurempia lukuja sitä varten on olemassa `long int` tyyppiset kokonaislukumuuttujat, joitka käyttävät kaksi sanaa. Tällöin suurin mahdollinen luku on $2^{63} - 1 = 9.22\ 10^{18}$.

Muuttuja `unsigned int` on etumerkitön kokonaisluku ja sen lukualue on $0, \ldots, 2^{32} - 1$.

Koska C- kielestä puuttuu looginen tietotyyppi, käytetään kokonaislukuja 0 ja 1 totuusarvojen "`epätosi`" ja "`tosi`" esittämiseen.

Reaaliluvut

Reaaliluvut voivat olla yksinkertaisen tarkuuden lukuja `float` tai kaksinkertaisen tarkuuden lukuja `double`. Yksinkertaisen takkuuden luvuissa on seitsemän merkitsevää numeroa ja kaksinkertaisen tarkkuden luvuissa 15 merkitsevää numeroa. Suurimmat mahdolliset luvut ovat vastaavasti 3.4×10^{38} ja 1.7×10^{308}.

Merkkitieto

Merkkitieto on tyyppiä `char`, ja kyseesssä on itseasiassa 8- bittinen kokonaisluku, joka vastaa tavua. Silloin merkkivakioiden arvot ovat välillä -128 ja 127 (`char`) tai välillä 0 ja 255 (`unsigned char`).

2.1.5 Vakiot

Vakioita on kolmea päätyyppiä: numeeriset vakiot, merkkivakiot ja merkkijonovakiot.

tyyppi	koko	suurin arvo	käyttö
unsigned char	8	255	merkki
char	8	127	merkki
int	32	2147483647	kokonaisluku
long int	64	9.22 $x10^{18}$	suuri kokonaisluku
float	32	3.4 $x10^{38}$	reaaliluku (yksinkertainen tarkkuus)
double	64	1.7 $x10^{308}$	reaaliluku (kaksinkertainen tarkkuus)

Numeeriset vakiot voivat olla joko kokonaislukuja tai desimaalilukuja, joissa kokonais- ja desimaaliosa on erotettu pisteellä. Desimaaliluvut voidaan esittää myös tieteellistä merkintätapaa käyttäen eli `0.0001 = 0.1e-3`.

Tyyppiä `long int` olevat **kokonaislukuvakiot** esitetään muodossa `123L`. Vastaavasti oktaalilukuvakiot esitetään muodossa `0123` ja hexadesimaaliluvut muodossa `0x123`. Siten esimerkiksi kymmenjärjestelmän luku `123` olisi oktaalilukuna `0173` ja heksadesimaalilukuna `0x7b`.

Merkkivakiot esitetään:

- Kirjoittamalla merkki heittomerkkeihin; esimerkiksi 'c'.
- Muodossa '\n', missä `n` on merkin ASCII- koodi oktaalilukuna; esimerkiksi '\007' on äänimerkki (BEL). Yleensä tätä tapaa käytetään vain ei- kirjoitettaville merkeille.
- Joillakin erikoistoiminnoilla on omat merkkinsä:
  - `'\n'` rivinvaihto (Line Feed)
  - `'\t'` tabulaattori (Horizontal Tab)
  - `'\v'` pystysuora tabulointi (Vertical Tab)
  - `'\b'` peruutusmerkki (Backspace)
  - `'\f'` sivunvaihto (Form Feed)
  - `'\\'` "kenoviiva"(Backslash)
  - `'\''` heittomerkki.

Merkkijonovakio on C- kielessä eri asia kuin yksittäinen merkki. Merkkijonot esitetään kirjoittamalla jonoon kuuluvat merkit lainausmerkkeihin; esimerkiksi `'A'` on merkkivakio kun taas "`A`" on merkkijonovakio. Merkkijonojen sisäinen esitys on erilainen kuin yksittäisten merkkien; itseasiassa merkkijonot tallennetaan taulukkoina, jonka alkiot ovat merkkejä ja viimeiseksi alkioksi lisätään NUL -merkki `'\0'`.

Esimerkkiksi merkkijonon "`Hello\n`" tallentamiseen tarvitaan 7 muistipaikkaa: yksi kullekin 5 merkistä `'H','e','l','l','o'` sekä yksi rivinvaihtomerkille `'\n'` ja lopuksi vielä yksi muistipaikka jonon loppumerkille `'\0'`.

2.1.6 Tulostus ja syöttö

Tulostus ruudulle ja syöttö näppäimistöltä eivät kuulu itse C- kieleen (kuten esimerkiksi BASICiin tai FORTRANiin) vaan ne suoritetaan erillisillä kirjastofunktioilla. Sitä varten kaikkissa esimerkeissä on otettu mukaan otsikkitiedosto `stdio.h`.

Direktiivi

```
#include <stdio.h>
```

kertoo C- kääntäjälle että tiedosto `stdio.h` otetaan mukaan käännökseen. Se sisältää muun muassa syöttö- ja tulostusfunktiot: `printf` ja `scanf`. Tiedoston tyyppi `.h` kertoo, että kysymyksessä on ns. **otsikkotiedosto** (header- file) ja C- kääntäjä osaa etsiä sen automaattisesti.

Tulostus

Ruudulle tulostetaan kuten edellä oli jo esillä funktiokutsulla

```
printf(muotoilu, arg1, arg2, ..., argn );
```

missä muotoiluosa on merkkijono, joka määrää tulostuksen ulkoasun ja argumentteja voi olla mielivaltainen määrä tai ei yhtään. Muotoiluosa rajoittimina käytetään siten lainausmerkkejä.

Muotoilu voi sisältää kenttämäärittelyjä, jotka alkavat prosentti- merkillä (`%`) jota seuraa joko yksi kirjain tai numero ja kirjain. Muut merkit tulostetaan sellaisinaan; myös välilyönnit ovat merkitseviä muotoilussa. Tärkeimpiä kentän määrittelykirjaimia ovat:

- `d` kokonaisluku
- `f` reaaliluku
- `e` reaaliluku, jolla on eksponentti
- `c` merkki
- `s` merkkijono

Jokaista argumettilistan muuttujaa on vastattava yksi kenttämäärittely muotoilussa. Tiedot tulostetaan mahdollisimman lyhyisiin kenttiin. Jos halutaan määrätä kentän koko ja desimaalien lukumäärä se kirjoitetaan prosentti- merkin ja määrittelykirjaimen väliin. Esimerkiksi muotoilu `%6d` tulostaa kokonaisluvun 6 merkin pituiseen kenttään ja muotoilu `%8.4f` tulostaa reaaliluvun 8 merkin pituiseen kenttään neljän desimaalin tarkkuudella.

Syöttö

Lukeminen tapahtuu funktiolla `scanf`, jonka kutsu on samanmuotoinen kuin `prinft-` funktiossa.

```
scanf(muotoilu, arg1, arg2, ..., argn );
```

Argumenttien edessä **on oltava** kuitenkin `&-` merkki. Tämä johtuu C- kielen käyttämästä funktion argumenttien välitysmekanismista; merkintä `&x` tarkoittaa sitä muuttujan `x` muistipaikan osoitetta, johon muuttujan `x` arvo luetaan (asiaan palataan luvussa 2.4). Syötettäessä tietoja näppäimistöltä syöttötiedot erotetaan toisistaan välilyönnillä.

Muotoilulle pätevät samat säännöt kuin printf käskyn muotoilulle.

Esimerkki tulostuksesta ja syötöstä

```
/* Esimerkkiohjelma area.c, jolla lasketaan ympyran ala.  Ohjelma
kysyy kayttajalta ympyran sadetta.  */

#include <stdio.h>

#define PI 3.14159

main()
{
   float r;
   float ala = 0.0;

   printf("\n%s\n%s",
      "Ohjelma laskee ympyran alan.",
```

```
      "Anna ympyran sade: ");
   scanf("%f", &r);

   ala = PI * r * r;

   printf("\n%s\n%s%.2f%s%.2f%s%.2f\n%s%8.4f\n",
       "A = PI * R * R ",
       "  = ",PI," * ",r," * ",r,
       "  = ", ala);
}
```

Kun ohjelma suoritetaan, saadaan seuraavanlainen tulostus:

```
Ohjelma laskee ympyran alan.
Anna ympyran sade: 3.0

A = PI * R * R
  = 3.14 * 3.00 * 3.00
  =  28.2743
```

2.1.7 Kommentit

Kommentit voidaan kirjoittaa merkkipareilla /∗ ja ∗/ erotettuina, jolloin kommentti kirjoitetaan niiden väliin; kommentti voi jatkua useammalle riville. Kommentin sisällä ei voi olla toista kommenttia.

2.1.8 Muuttujien alustaminen

Muuttujat voidaan **alustaa (initialisoida)** antamalla niille alkuarvot samalla kun ne määritellään. Esimerkiksi

```
#include <stdio.h>
main()
{
  float a1 = 1.0;
  float a2 = 2.0;
  float a3 = 3.0;
  float summa = 0.0;
  char  merkki = 'C';

  summa = summa + a1 + a2;
  summa = summa + a3;
  printf("%f %f %f %f \n", a1, a2, a3, summa);
  printf("%c\n", merkki);
}
```

Erityisesti jos muuttujan alkuarvon on oltava nolla (kuten tässä esimerkissä muuttujan `summa`), on se kätevintä alustaa määrittelyn yhteydessä. Ellei alustusta tehdä muuttujan arvo on *mielivaltainen*.

2.1.9 Symboliset vakiot

C- kääntäjissä on mukana preprosessori, joka käsittelee C- ohjelmatiedoston ennen sen kääntämistä. Silloin määrätyt merkkijonot muutetaan johonkin toiseen muotoon. Esimerkiksi voidaan määritellä `#define`-direktiivin avulla symbolisia vakioita muuttujille:

```
#include <stdio.h>
#define PI 3.141
main()
{
  float r, ala;

  r = 2.0;
  ala = PI * r * r;

  printf("Ympyran ala = %f \n", ala);
}
```

Preprosessori suorittaa silloin jo käännösvaiheessa vakion `PI` arvojen sijoitukset. Symbolisen vakion **käytön etuna** on se, että jos sitä joudutaan muuttamaan muutos tarvitsee tehdä vain yhteen kohtaan ohjelmassa.

2.2 Perusrakenneosat

C- ohjelmien rivirakenne on vapaa. Vaikka C- kielen määrittelyt eivät aseta rajoituksia, niin rivin pituutena on järkevää käyttää rivejä, jotka sopivat hyvin näyttöön (esimerkiksi korkeintaan 80 merkkiä).

Itse C- kielen syntaksi asettaa hyvin vähän vaatimuksia ohjelman ulkoasulle, mutta ohjelman luettavuuden kannalta on syytä noudattaa tiettyä vakiintunutta ohjelmointityyliä. Hyviä periaatteita ovat esimerkiksi

- Samassa tiedostossa olevat funktiot erotetaan ainakin yhdellä tyhjällä rivillä.
- Jokainen lause aloitetaan uudelta riviltä.
- Lause, joka on toisen lauseen osana, sisennetään.
- Ohjelmointirakenteet sisennetään.
- Usein sisennetään myös koko funktion runko.
- Symbolisten vakioiden nimet kirjoitetaan isoilla kirjaimilla.
- Käytetään suhteellisen lyhyitä rivejä.

2.2.1 Merkit ja symbolit

C- ohjelmissa voidaan käyttää seuraavia merkkejä:

- isot ja pienet kirjaimet väliltä a-z ja A-Z
- numerot 0-9
- välilyönti (blanko), tabulaattori, rivinvaihto, sivunvaihto
- erikoismerkkejä kuten , ; ( ) { } [ ] # % & ? + − ∗ / y.m.

Isoilla ja pienillä kirjaimilla on eri merkitys, joten esimerkiksi muuttuja `Summa` on eri asia kuin `summa`. Skandinaavisten kirjaimien käyttöä on syytä välttää niin muuttujissa kuin myös kommenteissakin. Tyhjää väliä (blanko, tabulaattori, rivinvaihto) saa olla vapaasti eri symbolien (tunnukset, varatut sanat, operaattorit) välissä (katso esimerkiksi edellä esitettyä ohjelmaa `area.c`).

2.2.2 Tunnukset ja muuttujat

Funktioiden ja muuttujien nimiä kutsutaan C- kielessä **tunnuksiksi** (identifiers).

- Tunnus saa sisältää kirjaimia (a-z ja A-Z), numeroita ja alaviivoja.
- Tunnuksen tulee alkaa kirjaimella tai alaviivalla.
- Tunnuksissa voi käyttää sekä isoja että pieniä kirjaimia, ja niiden ero on merkitsevä.
- Tunnuksen pituutta ei ole rajoitettu, mutta joissakin järjestelmissä vain 8 ensimmäistä merkkiä huomioidaan.
- Välilyöntejä ei voi käyttää tunnuksissa (usein käytetään sen sijaan alaviivaa "yhdyssanoissa"sanojen erottimena, esimerkiksi tyyliin `error_handling`).

Hyvää ohjelmointityyliä on valita tunnukset siten, että ne kuvaavat muuttujan tai funktion tehtävää.

2.2.3 Varatut sanat

Varattuja sanoja (keywords) ei saa käyttää funktioiden ja muuttujien niminä, vaan ne on varattu tiettyihin tarkoituksiin, kuten muuttujien tyypeiksi ym. Varattuja sanoja ovat muunmuassa:

auto	do	for	return	typedef
break	double	goto	short	union
case	else	if	sizeof	unsigned
char	enum	int	static	void
continue	extern	long	struct	while
default	float	register	switch	volatile

Varatut sanat kirjoitetaan aina pienin kirjaimin. Jos varattuja sanoja käytetään vahingossa esimerkiksi muuttujien tai funktioiden niminä saadaan aikaan hyvinkin kummallisia virheilmoituksia.

2.2.4 Aritmeettiset operaattorit

C- kielessä on kahdenlaisia operaattoreita: *Yhteen* operandiin kohdistuvia operaatoreita ovat esimerkiksi miinus- etumerkki sekä lisäys- ja vähennysoperaattorit (++ ja −−). Useimmat operaattorit kohdistuvat kahteen operandiin. Sellaisia ovat mm aritmeettiset operaattorit. C- kielen aritmeettiset operaattorit ovat: +, −, ∗, /, %.

Operaattori % on jakojäännös; esimerkiksi `10 % 3` =. Kokonaislukujen jakolaskussa tulos katkaistaan kokonaisluvuksi (ei pyöristetä); esimerkiksi `10 / 3 = 3` samoin kuin `11 / 3 = 3`.

Potenssiinkorotusta C- kielessä ei ole vaan se on tehtävä kirjastofunktiolla `pow`, joka sisältyy kirjastoon `math.h` (katso luku 2.6).

Esimerkki

```
/*  kokonaislukujen aritmetiikkaa  */

#include <stdio.h>
main()
{
  int     x,
          y,
          z;
  x = ( 3*2 )/( 9-4);
  y = -x;
  z = 7 % 2;
  printf("x=%3d y=%3d z=%3d\n",x,y,z);
}
```

Tulostus:

```
x =  1 y = -1 z =  1
```

2.2.5 Sijoitusoperaattorit

C- kielessä sijoitusoperaattorilla = muodostettu rakenne on myös lauseke. Esimerkiksi sijoituslauseessa `y = x = 1`, vakio `1` sijoitetaan ensin muuttujan `x` arvoksi, ja sen jälkeen sijoitusta käytetään lausekkeen `x = 1` arvona eli myös muuttujan `y` arvoksi sijoitetaan vakio `1`, `y = ( x = 1)`. Tavallisesti sijoituslausekkeen oikea puoli on jokin lauseke (kuten `x = a + b`).

C- kielen tavallisimmat sijoitusoperaattorit ovat: $=$ $+=$ $-=$ $*=$ $/=$

Neljä viimeistä operaattoria ovat lyhennysmerkintöjä usein esiintyville sijoituslauseille, jotka ovat muotoa

```
x op = y      on sama kuin   x = x op y
```

Esimerkiksi

```
 x += y       on sama kuin   x = x + y
 x *= 2       on sama kuin   x = x * 2
 x -= y + z   on sama kuin   x = x - (y + z)
 x /= y + z   on sama kuin   x = x / (y + z)
```

Huomaa, että koska sijoitusoperaattorit ovat assosiatiivisiä oikealta vasemmalle viimeisessä esimerkisssä muuttuja `x` jaetaan sijoituslauseen koko oikeanpuoleisella lausekeella `y+z`.

2.2.6 Lisäys- ja vähennysoperaattorit

Usein joudutaan käyttämään lausekkeita, jotka ovat muotoa `x = x + 1` tai `x = x - 1`. eli muuttamaan muistipaikassa x olevan luvun arvoa. C- kielessä tähän tarkoitukseen on olemassa omat operaattorinsa: $++$ ja $--$.

Lisäys- ja vähennysoperaattorit $++$ ja $--$ kasvattavat tai pienentävät muuttujan arvoa yhdellä:

- Jos operaattori $++$ on operandissa edessä, niin lisäys tapahtuu ennen operandin käyttöä laskennassa, joten lauseketta käytettäessä sen arvo on operandin arvo lisäyksen jälkeen.

- Jos operaattori $++$ on operandissa jäljessä, niin lisäys tapahtuu operandin käytön jälkeen, joten lausekketta käytettäessä sen arvo on operandin alkuperäinen arvo.

Vastaavat säännöt ovat voimassa myös vähennysoperaattorille.

Esimerkki selventänee näiden operattorien käyttöä parhaiten. Olkoon muuttuja `k` tyyppiä `int` ja sen arvo olkoon `k = 7`, silloin

Lauseke	k:n arvo ennen lausekkeen laskua	k:n arvo lausekkeen laskun jälkeen	Lausekkeen arvo
$++k+3$	8	8	11
$--k+3$	6	6	9
$3+k++$	7	8	10
$3+k--$	7	6	10

Esimerkki

```
/* Lisäys- ja vähennysoperaattorit */

#include <stdio.h>
main()
{
        int     i,
                j,
                k;
        i = 3;
        k = i++;
        printf("k = %2d i = %2d \n", k, i );
        k = ++i;
        printf("k = %2d i = %2d \n", k, i );
        i--;
        j = --i;
        printf("j = %2d i = %2d \n", j, i );
}
```

Tulostus:

```
k =  3 i =  4
k =  5 i =  5
j =  3 i =  3
```

2.2.7 Sidontajärjestys ja assosiointi

Jos lausekkeessa on useita operaattoreita, niin kääntäjän täytyy tietää, mihin operandeihin kukin operaattori kohdistuu. Tätä varten on olemassa **sidontasäännöt** (priority, precedence). Sidontajärjestys voidaan osoittaa myös sulkumerkeillä. Esimerkiksi

```
a + b * c    on sama kuin a + (b * c),
```

koska kertolasku suoritetaan ennen yhteenlaskua, eli kertolaskuoperaattorin presedenssi on suurempi kuin yhteenlaskun

Monisteen Liitteessä B on esitetty C- kielen operaattorien sidontajärjestys.

C- kielessä useimmille operaattoreille operandin liittäminen tapahtuu vasemmalta oikealle, eli tasaarvoisten operaattoreiden tapauksessa laskujärjestys on vasemmalta oikealle. Poikkeuksena ovat yhteen operandiin kohdistuvat operaattorit ja sijoitusoperaattorit ($=$ ja $+=$, $-=$, $*=$, $/=$), joissa liittäminen tapahtuu oikealta vasemmalle. Tämä tarkoittaa sitä, että

```
a / b * c    on sama kuin   (a / b) * c
a = b = c    on sama kuin   a = (b = c)
a /= b + c   on sama kuin   a = a / (b + c)
```

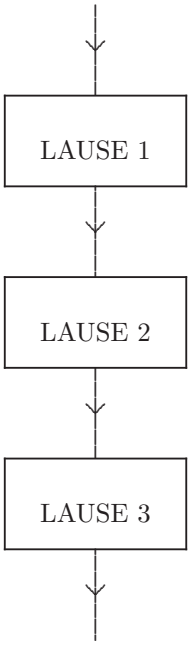
2.3 Ohjausrakenteet

Ohjelmointikielissä lauseiden suoritusjärjestykseen liittyviä perusrakenteita on kolme:

- Jono
- Valinta
- Toisto

2.3.1 Jono

Jono muodostuu lauseista, jotka suoritetaan peräkkäin siinä järjestyksessä, missä ne on kirjoitettu (katso kuva 2.1. Nämä lauseet voidaan ryhmittää erilliseksi **lohkoksi** (blokki, block), joka muodostuu aaltosulkujen ({ ja }) väliin alekkain kirjoitetuista määrittelyistä lauseista. Jos lohkon sisällä tehdään määrittelyjä, ne ovat voimassa vain kyseisen lohkon loppumerkkiin saakka. Funktion runko on esimerkki lohkosta.



Kuva 2.1: Peräkkäinen lauseiden suoritusjärjestys

C- kielen lauseet suoritetaan normaalisti peräkkäin. Usein kuitenkin halutaan valita vain jotkut lauseet suoritettavaksi tai toistaa määrättyjä lauseita useamman kerran. Näitä tehtäviä varten on olemassa **valinta- ja toistolauseet**. Ehdot, joiden perusteella valinnat tehdään muodostetaan vertailuoperaattoreiden ja/tai loogisten operaattoreiden avulla.

2.3.2 Valintalauseet: `if-else, switch`

Usein tulee vastaan tilanteita, joissa halutaan valita eri vaihtoehtojen – lohkojen – välillä. Tällöin täytyy käyttää ohjelmointikielen valintarakenteita. Seuraavassa esitellään kaksi tapaa. Ensimmäinen perustuu `if-` lauseeseen, jossa valinta tapahtuu kahden vaihtoehdon välillä. Tätä rakennetta voi laajentaa monivalintaan siten, että toinen vaihtoehto johtaa uuteen valintalauseeseen. Toisena rakenteena esitellään `switch`-lause, jota käytetään silloin, kun valittavana on useita vaihtoehdoja.

`if-` lause

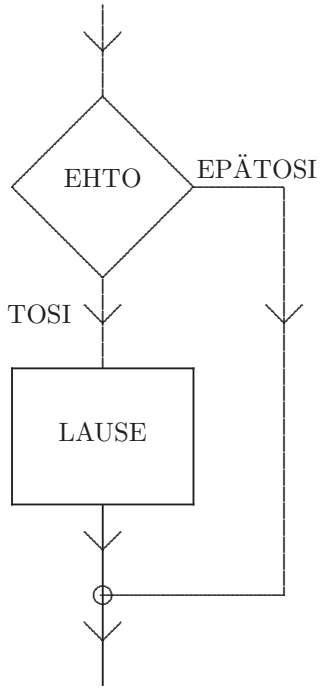
C- kielessä `if-` lauseen yleinen muoto on

```
if (ehto)
  lause
```

missä lause voi olla joko yksittäinen lause tai usean peräkkäisen lauseen muodostama lohko.

Jos `ehto` on tosi `lause` suoritetaan, muussa tapauksessa se ohitetaan.

Vaihtoehtoiset tilanteet ovat useimmiten monivalintaisia, jolloin on käytettävä `if- else-` rakennetta:



Kuva 2.2: `if`- rakenne

```
if (ehto)
  lause 1
else
  lause 2
```

Jos `ehto` on tosi `lause 1` suoritetaan ja `lause 2` ohitetaan, muussa tapauksessa (eli `ehto` on epätosi) `lause 1` ohitetaan ja `lause 2` suoritetaan. Sen jälkeen suoritus jatkuu `if-else`- rakennetta seuraavasta lauseesta.

Ehtolauseet, joiden perusteella valinta tapahtuu, muodostetaan vertailuoperaattoreiden ja loogisten operaattoreiden avulla.

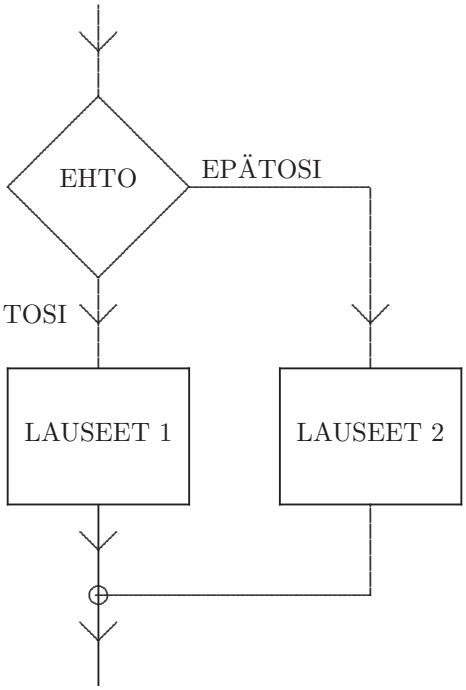
Vertailuoperaattorit

Vertailuoperaattori antaa tulokseksi ykkösen (1) tai nollan (0) sen mukaan, onko kyseinen relaatio tosi vai epätosi.

C- kielen vertailuoperaattoreita ovat:

symboli	merkitys	matemaattinen symboli
>	suurempi kuin	$>$
>=	suurempi tai yhtäsuuri kuin	$\geq$
<	pienempi kuin	$<$
<=	pienempi tai yhtäsuuri	$\leq$
==	yhtäsuuri kuin	$=$
!=	erisuuri kuin	$\neq$

Vertailuoperaattoreiden operandien tulee olla lukuja, merkkejä tai osoitteita. **Merkkijonojen** vertailua **ei voida** suorittaa vertailuoperattoreilla vaan siihen on käytettävä erityisiä merkkijonofunktioita (katso 2.9).



Kuva 2.3: `if-else`- rakenne

Huomaa, että **yhtäsuuruusvertailuun** käytetään C- kielessä siis **kahta** yhtäsuuruusmerkkiä (==) kun taas **yksi** yhtäsuuruusmerkki (=) on sijoitusoperaattori.

Loogiset operaattorit

C- kielen loogiset operaattorit ovat: **ja** (`&&`), **tai** (||) sekä **ei** (`!`).

Loogisten operaattorien operandit voivat olla mitä tahansa kokonaislukuja: nolla (0) esittää totuusarvoa `epätosi` ja mikä tahansa muu luku (esimerkiksi 1) esittää totuusarvoa `tosi`.

Negaatio (`not, ei`) muuttaa totuusarvon eli antaa tulokseksi ykkösen (tosi), jos operandi on nolla (0) ja nollan (epätosi), jos operandi on mikä tahansa muu luku; esimerkiksi:

e	!e
0	1
1	0
10	0

Operaattorien `ja (and)` ja `tai (or)` totuustaulut ovat:

e1	e2	e1 && e2	e1 || e2
F	F	F	F
F	T	F	T
T	F	F	T
T	T	T	T

missä F (false) merkitsee epätosi (0) ja T (true) merkitsee tosi (1 tai suurempi).

Esimerkkejä `if`-lauseesta

• Lasketaan luvun itseisarvo:

```
main()
{
  float a, abs;

  printf("Anna luku: ");
  scanf("%f", &a);

  if (a > 0) {
    abs = a;
  }
  else {
    abs = -a;
  }
  printf("Itseisarvo on: %f\n", abs);
}
```

- Tutkitaan onko luettu merkki pieni kirjain.

```
main()
{
  char merkki;

  printf("Kirjoita kirjain: ");
  scanf("%c", &merkki);

  if ('a' <= merkki && merkki <= 'z') {
    printf("%c on  pieni kirjain\n)", merkki);
  }
  else {
    printf("%c ei ole  pieni kirjain\n)", merkki);
  }
}
```

- Tulostetaan kolme lukua suuruusjärjestyksessään.

```
#include <stdio.h>
main()
{
 int     i, j, k;

 printf("anna kolme kokonaislukua\n");
 scanf("%d%d%d", &i, &j, &k );
 printf("luvut suuruusjärjestyksessä\n");

 if (i>j && i>k) {
    if (j>k)
       printf("%d,%d,%d\n",i,j,k);
    else
       printf("%d,%d,%d\n",i,k,j);
  }
  else if (j>k) {
    if (i>k)
       printf("%d,%d,%d\n",j,i,k);
```

```
    else
       printf("%d,%d,%d\n",j,k,i);
  }
  else {
    if (i>j)
       printf("%d,%d,%d\n",k,i,j);
    else
       printf("%d,%d,%d\n",k,j,i);
  }
}
```

Tulostus:

```
anna kolme kokonaislukua
4 6 1
luvut suuruusjärjestyksessä
6,4,1
```

Haarautuminen: `switch`-lause

Ehtorakenteessa voi olla suoritettavien lauseiden paikalla toinen ehtolause (joko yksinkertainen `if`- lause tai `if-else`- lause) eli niinsanottu "nested `if`- statement". Usein valintatilanne on kuitenkin sellainen, että halutaan suorittaa valinta useiden vakiovaihtoehtojen välillä. Tähän tarkoitukseen on C- kielessä olemassa `switch`- rakenne:

```
switch(valintalauseke)
{
case vakio1:
  lause 1;
  break;
case vakio2:
  lause 2;
  break;
  .
  .
  .
default:
  lause n+1;
}
```

Ensin lasketaan `valintalausekkeen` arvo, ja sen perusteella suoritus jatkuu jollakin `case`- lausekkeella, jos `valintalausekkeen` arvo on sama kuin jokin vakioista (`vakio1`,...,`vakion`) ja muussa tapauksessa suoritetaan lauseke `default`. Jos `break` lauseke puuttuu suoritetaan seuraavakin `case`- lauseke.

Joissakin tapauksissa `switch`- lause voidaan esittää myös `if- else if`- rakenteen avulla. Tällöin ehtolausekkeet täytyy muodostaa valintalausekkeen ja vakioiden 1...n avulla; esimerkiksi muodossa `valintalauseke==vakio1`.

```
if (tosi1) {
  lause 1;
}
else if(tosi2) {
  lause 2;
  .
  .
```

```
  .
}
else {
  lause n+1;
}
```

Tyypillinen esimerkki `switch`- lauseen käytöstä on valikko (menu):

```
int i;
printf("Suorita valinta:\n");
printf("(0 = editoi, 1 = tulosta, 2 = tuhoa, 3 = lopeta) ");

scanf("%d", &i);
switch(i) {
case 0:
  edit();
  break;
case 1:
  print();
  break;
case 2:
  delete();
  break;
case 3:
  end();
  break;
default:
  error();
}
```

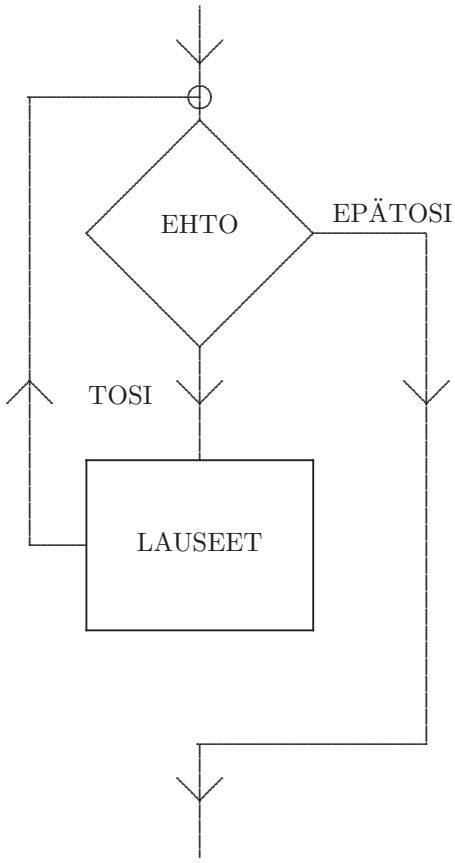
Esimerkki, jossa ohjelman käyttäjä voi valita kahdelle luvulle suoritettavan aritmeettisen operaation.

```
/*  Laskutoimituksen merkki */
#include <stdio.h>
main()
{
  char    merkki;
  float   a, b, tulos;
  printf("anna kaksi lukua \n ");
  scanf("%f%f", &a, &b);
  printf("anna laskutoimitus\n");
  scanf("\n%c", &merkki);
  switch  (merkki) {
    case '+':
      tulos   = a + b;
      break;
    case '*':
      tulos   = a * b;
      break;
    case '-':
      tulos   = a - b;
      break;
    case '/':
```

```
      tulos   = a / b;
      break;
    default:
      printf("väärä merkki \n");
      break;
  }
  printf("Vastaus:\n %5.2f%2c%5.2f=%5.2f \n",
         a, merkki, b, tulos );
}
```

Ohjelma antaa seuraavanlaisen tulostuksen.

```
anna kaksi lukua
 4.2 5.3
anna laskutoimitus
/
Vastaus:
  4.20 / 5.30= 0.79
```



Kuva 2.4: `while`- rakenne

2.3.3 Toistolauseet: `while, do, for`

C- kielessä on kolme toistolausetta: `while`-, `do`- ja `for`- lauseet.

`while`- lause

Toistorakenne `while` on muotoa:

```
while (ehto) {
  lauseet;
}
```

Rakenteen suoritus alkaa `ehto`- lausekeen laskemisella. Jos `ehto` on tosi `lause` suoritetaan, jonka jälkeen `ehto` lasketaan uudelleen ja suoritetaan sen jälkeen taas uudelleen `lause`, mikäli `ehto` oli vieläkin tosi. `while`- lauseen suoritus päättyy kun `ehto` on epätosi. Siten on mahdollista, ettei lausetta suoriteta kertaakaan.

Esimerkkejä

```
/*      Keskiarvon laskeminen  */

#include <stdio.h>
main()
{
  int     i = 0;
  float   summa = 0.0,
          keski,
          x;

  printf("anna lukuja, lopuksi > 1.e20\n");
  scanf("%f", &x );

  while ( x < 1.e20) {
     summa += x;
     i++;
     scanf("%f", &x );
  }
  keski   = summa / i;

  printf(" keskiarvo = %5.2f\n",keski);
}
```

Toisena esimerkkinä muutetaan tekstistä pienet kirjaimet isoiksi kirjaimiksi:

```
/* pienet kirjaimet isoiksi */

#include <stdio.h>

main()
{
  int c;

  while ((c = getchar() ) != EOF){
     if (c >= 'a' && c <= 'z') {
putchar(c - 'a' + 'A');
     }
     else {
putchar(c);
     }
  }
}
```

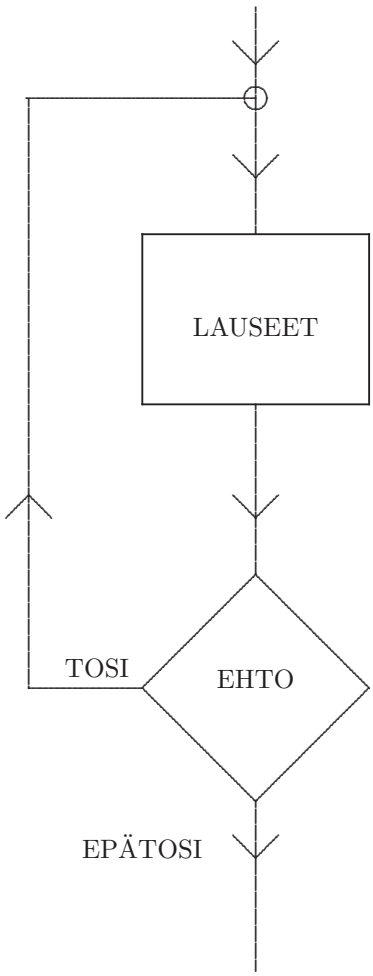
Funktio `getchar` lukee yhden merkin kerrallaan standardisyötöstä (eli näppäimistöltä) ja funktio `putchar` tulostaa yhden merkin standardiulostuloon (eli näytölle). `while`- lauseen ehtolauseessa luetaan ensin merkki ja testataan, onko se tiedoston loppumerkki (symbolinen vakio `CTRL D`). Jos on luettu jokin muu merkki kuin `EOF`, testataan seuraavaksi kuuluuko merkin ASCII- koodi välille (`a,...,z`): eli se on pieni kirjain. Jos kuuluu muutetaan merkki isoksi vähentämällä ensin merkin `a` ASCII- koodi ja lisäämällä sen jälkeen merkin `A` ASCII- koodi ennen tulostusta, mutta muussa tapauksessa tulostetaan merkki sellaisenaan.

`do`- lause

Toistorakenne `do` on muotoa:

```
do {
  lauseet;
} while (ehto)
```

`do`- lauseessa `lause` suoritetaan ainakin kerran (toisin kuin `while`- lauseessa), koska `ehto` testataan vasta sen jälkeen, kun `lause` on jo suoritettu. Jos `ehto` on tosi, niin toistoa jatketaan suorittamalla `lause` uudelleen, muussa tapauksessa `do`- lauseen soritus lopetetaan. **Esimerkkejä** `do`- lauseen käytöstä.



Kuva 2.5: `do`- rakenne

Luetaan lukuja niin kauan, kunnes tulee ensimmäinen negatiivinen luku:

```
int n;
do {
scanf("%d", &n);
} while(n >= 0)

/* suurimman luvun etsiminen */
```

```
#include <stdio.h>
main()
{
  int     suurin = -32000;
  int     i;

  printf("anna kok.lukuja, lopuksi -32000\n");
  do {
      scanf("%d",&i);
      if(i>suurin) {
         suurin  = i;
      };
   } while(i!=-32000);
   printf("suurin luku oli %d \n",suurin);
}
```

`for`- lause

Toistorakenne `for` on muotoa:

```
for (alkulauseke; ehtolauseke; toistolauseke) {
   lauseet;
}
```

Mikä tahansa lausekkeista (`alku`-, `ehto`- tai `toistolauseke`) voi puuttua, mutta sulut ja puolipisteet on kirjoitettava siinäkin tapauksessa.

`for`- lauseen lausekkeiden suoritusjärjestys on seuraava:

- lasketaan ensin `alkulauseke` eli silmukan alustus
- suoritetaan itse toistosilmukka, missä kullakin kierroksella
  - lasketaan ehtolauseke ja jos tulos on tosi jatketaan silmukan suoritusta, muussa tapauksessa poistutaan `for`- lauseesta
  - suoritetaan `lause`
  - lasketaan `toistolauseke`, mikä hyvin usein tarkoittaa kierroslaskurin inkrementointia.

`for`- lause voidaan esittää myös `while`- lauseen avulla muodossa:

```
alkulauseke;
while(ehtolauseke){
  lauseet;
  toistolauseke;
}
```

Jos `alku`- tai `toistolauseke` puuttuvat, niitä vastaavat lauseet jäävät pois `while`- rakenteesta. Jos taas `ehtolauseke` puuttuu, se tulkitaan todeksi.

Esimerkkinä `for`- lauseen käytöstä tulostetaan kaikkien pienten kirjainten ASCII- koodit:

```
/* esimerkki for -lauseesta */
#include <stdio.h>

main()
{
  int c;
```

```
  printf("merkki \t ASCII- koodi\n");

  for (c = 'a'; c <= 'z'; c++) {
     printf("%c \t %d\n", c, c);
  }
}
```

2.3.4 Tyhjä lause

Tyhjä lause on sisällöltään tyhjä ja sisältää vain puolipisteen; se ei tee mitään. Tyhjää lausetta tarvitaan kun C- kielen säännöt vaativat jossakin rakenteessa lauseen olemassaoloa, mutta siinä kohtaa ei kuitenkaan haluta tehdä mitään.

Tyypillinen esimerkki olisi luettujen merkkien lukumäärän laskeminen yhtäaikaa niiden lukemisen kanssa:

```
/* esimerkki: lasketaan  merkkien lukumaara */
#include <stdio.h>
main()
{
  int nc;

  for (nc=0; getchar() != EOF; ++nc){
     ;
  }
  printf("Merkkien lukumaara = %d\n", nc);
}
```

2.3.5 Esimerkkejä

Esimerkki 1, etsitään suurin luku päätteeltä luetuista reaaliluvuista:

```
/* esimerkki maximum.c */
#include <stdio.h>
main()
{
  int i, n;
  float max, x;

  printf("Montako lukua luetaan? " );
  scanf("%d", &n);

  while (n <= 0) {
    printf("\nVIRHE: anna positiivinen luku.\n");
    printf("Montako lukua luetaan? " );
    scanf("%d", &n);
  }

  printf("\n Anna %d reaalilukua: ",n);
  scanf("%f", &x);
  max = x;
  i = 1;
  while(i < n) {
```

```
    scanf("%f", &x);
    if (max < x){
      max = x;
    }
    ++i;
  }
  printf("\nSuurin luku = %f\n\n", max);
}
```

Kun ohjelma suoritetaan, saadaan seuraavanlaisia tulosteita:

```
Montako lukua luetaan? -1

VIRHE: anna positiivinen luku.
Montako lukua luetaan? 3

Anna 3 reaalilukua: 1.0 3.0 10

Suurin luku = 10.000000
```

Esimerkki 2, lasketaan luvun kertoma kaikilla kolmella toistorakenteella:

```
/* Esimerkki while- silmukasta */
#include <stdio.h>

main()
{
  int lukumaara;
  int luku = 1;
  long int kertoma = 1;

  printf("\nMontako kertomaa lasketaan: ");
  scanf("%d", &lukumaara);

  printf("%11s %13s\n", "Luku (n)", "Kertoma (n!)");
  while ( lukumaara >=  luku) {
    kertoma *= luku;
    printf("%9d %12ld\n", luku, kertoma);
    luku++;
  }
}
```

```
/* Esimerkki do-silmukasta */
#include <stdio.h>

main()
{
  int lukumaara;
  int luku = 1;
  long int kertoma = 1;

  printf("\nMontako kertomaa lasketaan: ");
  scanf("%d", &lukumaara);
```

```
  printf("%11s %13s\n", "Luku (n)", "Kertoma (n!)");

  do {
    kertoma *= luku;
    printf("%9d %12ld\n", luku, kertoma);
    luku++;
  } while ( lukumaara >=  luku);
}

/* Esimerkki for- silmukasta */
#include <stdio.h>

main()
{
  int lukumaara;
  int luku = 1;
  long int kertoma = 1;

  printf("\nMontako kertomaa lasketaan: ");
  scanf("%d", &lukumaara);

  printf("%11s %13s\n", "Luku (n)", "Kertoma (n!)");
  for (luku = 1; lukumaara >=  luku; luku++) {
    kertoma *= luku;
    printf("%9d %12ld\n", luku, kertoma);
  }
}
```

Kaikkien kolmen ohjelman tulostus olisi seuraavanlainen.

```
Montako kertomaa lasketaan: 10
  Luku (n)  Kertoma (n!)
        1            1
        2            2
        3            6
        4           24
        5          120
        6          720
        7         5040
        8        40320
        9       362880
       10      3628800
```

2.4 Funktiot

Modulaarinen ohjelmointi on menetelmä ohjelmointiongelmia ratkaistaessa. Sen kaksi pääperiaatetta ovat:

- Ohjelman ohjausrakenteet pidetään mahdollisimman yksinkertaisina.
- Ohjelmiston suunnittelussa noudatetaan ”top- down design” menetelmää.

Top- down design tarkoittaa ongelman jakamista astettain pienempiin ja helpommin ratkaistaviin palasiin (stepwise refinement). Lopulta ongelma on jaettu osatehtäviin, joista kukin voidaan suorittaa yhdella funktiolla, ja jotka ovat lisäksi helposti ohjelmoitavissa.

C- kielen funktiomekanismi on soveltuu hyvin näiden alimman tason erillisten ohjelmointitehtävien ratkaisemiseen. Itse ohjelma kootaan näista funktiosta ja lopuksi `main`- funktio suorittaa alkuperäisen tehtävän.

2.4.1 Funktioiden kutsut

Ohjelman suoritus alkaa funktiosta `main`. Kun ohjelman suorituksessa kutsutaan jotain funktiota, ohjelman hallinta siirtyy sille. Kun kutsuttu funktio on suorittanut sille suunnitellut tehtävät, ohjelman suoritus palaa kutsun suorittaneelle funktiolle ja lopulta funktiolle `main`, josta poistuttaessa ohjelman suoritus päättyy.

Esimerkki yksinkertaisesta funktiokutsusta:

```
#include <stdio.h>
void prn_message();

main()
{
          prn_message();
}

void prn_message()
{
        printf(''Hello World\n'');
        printf(''Have a nice day\n'');
}
```

2.4.2 Funktion määrittely

Funktion yleinen määrittely on muotoa:

```
tyyppi nimi(argumenttilista määrittelyineen)
{
  funktion sisäiset määrittelyt
  lauseet

}
```

Tyyppi kertoo funktion tyypin, jotka ovat samat kuin muuttujien tyypit (katso luku 2.2); jos tyyppiä ei ole määritelty, sen oletusarvo on `int`. Jos funktio ei palauta mitään arvoa sen tyypiksi voidaan määritella `void`. Argumenttilista voi olla myös tyhjä, mutta sulkumerkit täytyy kirjoitaa siinäkin tapauksessa.

2.4.3 Funktion paluuarvo

Funktion arvo palautetaan `return`- lauseella. Sillä on kaksi tarkoitusta:

- Kun `return`- lause suoritetaan, funktio palauttaa heti kontrollin kutsuvalle funktiolle
- `return`- tunnusta seuraavan lausekkeen arvo palautetaan funktion arvona; tämän lausekken arvon on oltava samaa tyyppia kuin funktiolle määritelty tyyppi.

Esimerkkinä valitaan kahdesta luvusta pienempi:

```
#include <stdio.h>
int min(int x, int y);

main()
{
   int j, k, m;

   printf("Anna 2 lukua. ");
   scanf("%d%d", &j, &k);
   m = min(j, k);
   printf("\n %d on pienempi luvuista %d ja %d\n", m, j, k);
}

int min(int x, int y)
{
  if (x < y){
    return(x);
  }
  else {
    return(y);
  }
}
```

2.4.4 Funktion argumentit

Funktion argumentit ovat **muodollisia argumentteja**. Ne saavat arvonsa siten, että funktion kutsun vastaavien **todellisten argumenttien** arvot sijoitetaan niihin. Tämä sijoitus on vain yhteen suuntaan. Toisin sanoen funktio voi muuttaa muodollisen argumenttinsa arvoa, mutta **todellisen argumentin arvo ei muutu**. Funktion muodollinen argumentti vastaa siten paikallista muuttujaa.

ANSI- standardin mukaan funktion muodollisten argumenttien tyyppimäärittelyt suoritetaan funktion otsikkorivillä argumenttilistassa.

Paikalliset muuttujat määritellään funktion rajaavien aaltosulkujen sisällä.

Esimerkki:

```
/* summan laskeminen */

#include <stdio.h>
int summaa(int n);      /* funktion prototyyppi */

main()
{
  int n=3, sum;

  printf("%d\n", n);
  sum = summaa(n);     /* funktion kutsu */
  printf("%d\n", n);
  printf("%d\n", sum);

}

int summaa(int n)       /* funktion määrittely */
{
```

```
  int sum = 0;

  for (; n>0; --n){
    sum += n;
  }
  printf("%d\n", n);
  return (sum);
}
```

Tulostus:

```
3
0
3
6
```

Huomaa, etta vaikka muuttujan `n` arvo muuttuu funktiossa `summaa`, se ei vaikuta muuttujan `n` arvoon funktiossa `main`, koska vain muuttujan arvo kutsuhetkellä välitetään funktiolle. Tämä argumenttienvälitysmekanismi (”call by value”) eroaa joissakin muissa kielissä kuten FORTRANissa käytetystä mekanismista, jossa itse argumenttien osoite välitetään funktiolla (”call by reference”).

C- kielessä tähän tarkoitukseen on olemassa pointterit (osoittimet), joita käsitellään luvussa (2.7). Itseasiassa tämä on syy siihen miksi `scanf`- funktion kutsussa argumentin eteen lisätään `&`- merkki sen merkiksi, että funktiolle onkin tässä tapauksessa välitetty muuttujan osoite. Tällä tavoin muuttujan arvoa voidaan muuttaa kutsuttavan funktion sisällä siten, että muutos välittyy myös kutsuvalle funktiolle.

2.4.5 Funktion prototyypit

Jos ohjelmatiedostossa esiintyy funktion kutsu, mutta itse funktio on toisessa tiedostossa tai sen koodi on samassa tiedostossa vasta itse funktion kutsun jälkeen, on funktio määriteltävä ennen kutsua. Funktion määrittelyssä voidaan kuvata sen argumenttilista funktion prototyypillä. Silloin kääntäjä voi tarkistaa muodollisten argumenttien tyypin kutsuvassa funktiossa. Argumettilista voidaan kuvata prototyypissä joko luettelemalla vain argumenttien tyypit tai ilmoittamalla lisäksi myös argumenttien nimet. Jos funktiolla ei ole argumentteja voidaan argumenttilistana käyttää varattua sanaa `void`.

2.4.6 Modulaarisuus

Useimmat käytännössä eteen tulevat ohjelmointitehtävät ovat niin laajoja, että ne on järkevää jakaa osiin, **moduleihin** ja käyttää top- down suunnittelumetodia. Moduli voi koostua yhdestä funktiosta tai siihen voi olla koottuna läheisesti toisiinsa liittyvät funktiot ja määrittelyt.

Esimerkkinä tehdään ohjelma, joka lukee lukuja niin kauan, kunnes kohdataan tiedoston loppumerkki, ja tulostaa sen jälkeen siihen mennessä luetuista luvuista suurimman ja pienimmän yhdessä itse luvun ja siihen mennessä luettujen lukujen summan kanssa. Ohjelman yleinen rakenne voisi silloin olla seuraava:

- tulosta otsikko
- tulosta otsakkeet
- laske ja tulosta tiedot

Ohjelman suunnittelun ensimmainen vaihe voisi olla:

```
/* Esimerkkiohjelma sminmax.c */
#include <stdio.h>
```

```
void prn_banner();
void prn_header();
void read_and_prn_data();

main()
{
  prn_banner();
  prn_header();
  read_and_prn_data();
}

void prn_banner()
{
  printf("\n\t%s\n\t%s\n\t%s\n",
 "****************************************",
 "* Lasketaan summat, mimimit ja maksimit *",
 "****************************************");
}

void prn_header()
{
  printf("prn_header ei kaytossa\n");
}

void read_and_prn_data()
{
  printf("read_and_prn-data ei kaytossa\n");
}
```

jolloin tuloste olisi:

```
****************************************
* Lasketaan summat, mimimit ja maksimit *
****************************************
prn_header ei kaytossa
read_and_prn-data ei kaytossa
```

Toinen vaihe voisi olla kahden viimeisen funktion kirjoittaminen:

```
void prn_header()
{
  printf("%12s%12s%12s%12s%12s\n\n",
    "J_numero", "Luku", "Summa", "Minimi", "Maksimi");
}

void read_and_prn_data()
{
   int i = 0, luku, summa, pienin, suurin;

   if(scanf("%d", &luku) == 1) {
     ++i;
     summa = pienin = suurin = luku;
     printf("%12d%12d%12d%12d%12d\n",
```

```
          i, luku, summa, pienin, suurin);
      while(scanf("%d", &luku) == 1) {
        ++i;
        summa += luku;
        pienin = min(luku, pienin);
        suurin = max(luku, suurin);
        printf("%12d%12d%12d%12d%12d\n",
           i, luku, summa, pienin, suurin);
    }
  }
  else
    printf("VIRHE: yhtaan lukua ei ole luettu.\n\n");
}
```

Kolmas vaihe voisi olla max ja min funktioiden kirjoittaminen, jolloin prototyypit

```
int max(int x, int y)

int min(int x, int y)

lisätään main-funktiota ennen.

/* suurempi luvuista x ja y       */
int max(int x, int y)
{
  if (x > y){
    return(x);
  }
  else {
    return(y);
  }
}

/* pienempi luvuista x ja y        */

int min(int x, int y)
{
  if (x < y){
    return(x);
  }
  else {
    return(y);
  }
}
```

Tämä on ohjelmointitehtävän lopullinen ratkaisu tässä tapauksessa. Itse asiassa tämä ohjelma toimii hyvin vain, kun datat luetaan DOSin syötön uudelleenohjauskomennolla, joten suunnittelua olisi jatkettava vielä ainakin yksi vaihe lisää, jos halutaan, ettei syöttö ja tulostus sotke näyttöä, kun luvut luetaan suoraan päätteltä.

Ohjelma voidaan suorittaa kirjoittamalla ensin luvut tiedostoon `koe.dat` ja käyttämällä sen jälkeen syötön uudelleenohjausta tiedostosta:

```
C> a.out < koe.dat
```

```
****************************************
* Lasketaan summat, mimimit ja maksimit *
****************************************
    J_numero        Luku       Summa      Minimi     Maksimi

           1           2           2           2           2
           2           4           6           2           4
           3           6          12           2           6
           4          11          23           2          11
           5          24          47           2          24
```

2.5 Merkkien käsittely

Tässä luvussa perehdytään merkkien käsittelyyn sekä myös merkkien käsittelyssä käytettäviin kirjastofunktioihin (erityisesti `putchar` ja `getchar`- funktioihin).

2.5.1 Merkkitietotyyppi

Merkkitietotyyppi `char` vie yhden tavun muistia eli 8 bittiä. Silloin erilaisia merkkejä voi olla $2^8 = 256$ kappaletta.

Merkit tallennetaan muistiin erityisen koodin, ASCII- koodin, avulla koodattuina. Merkkimuuttujat ovat silloin itseasiassa ASCII- koodiaan vastaavia pieniä kokonaislukuja, joilla voidaan suorittaa myös aritmeettisia operaatioita kuten kokonaisluvuilla. Kaikki merkit eivät ole kirjoitettavia (esimerkiksi sivun vaihto, äänimerkki y.m.). Liitteessä A on esitetty printtautuvien merkkien ASCII- koodit.

Esimerkkinä soitetaan koneella äänimerkkiä n kertaa:

```
#include<stdio.h>

main()
{
  int i, n;

  printf("Montako aanimerkkia? ");
  scanf("%d", &n);

  for(i=1; i<= n; ++i) {
        printf("%c", '\007');
  {
}
```

2.5.2 Funktiot `getchar` ja `putchar`

Funktiolla `getchar` luetaan merkkejä merkki kerrallaan standardisyötöstä (näppäimistöltä) rivinsiirtoon (ENTER) saakka ja funktiolla `putchar` tulostetaan yksi merkki standardiulostuloon (näyttöön). Syöttö voidaan tietenkin ohjata tulemaan myös tiedostosta ja vastaavasti tulostus ohjata tiedostoon DOSin uudelleenohjauskomennoilla (< `ja` >).

2.5.3 Merkkifunktiot

C-kieleen kuuluu otsikkotiedosto (header- file) `ctype.h`, joka sisältää paljon muitakin merkkienkäsittelyfunktioita kuin `getchar` ja `putchar` (useimmat ovat itseasiassa makroja).

Makrot, jotka suorittavat testejä palauttavat arvonaan nollan (0, epätosi) tai ei- nollan (1, tosi):

isalpha(c)	c is a letter
isupper(c)	c is an upper case letter
islower(c)	c is a lower case letter
isdigit(c)	c is a digit
isxdigit(c)	c is a hex digit
isalnum(c)	c is an alphanumeric character
isspace(c)	c is a space, tab, carriage return, newline, or formfeed
ispunct(c)	c is a punctuation character (neither control nor alphanumeric)
isprint(c)	c is a printing character code 040(8) (space) through 0176 (tilde)
isgraph(c)	c is a printing character similar to isprint except false for space.
iscntrl(c)	c is a delete character (0177) or ordinary control character (less than 040).
isascii(c)	c is an ASCII character, code less than 0200

Seuraavat makrot suorittavat merkkien konversioita:

tolower(c)	c is converted to lower case. Return value is undefined if not isupper(c).
toupper(c)	c is converted to upper case. Return value is undefined if not islower(c).
toascii(c)	c is converted to be a valid ASCII character.

2.5.4 Esimerkki

Esimerkkinä tarkastellaan sanojen lukumäärän laskemista, kun niitä kirjoitetaan näppäimistöltä. Käytetään jälleen top- down menetelmää ja jaetaan ongelma pienempiin osiin:

- sanan määrittely: sanat ovat merkkijonoja, jotka erotetaan toisistaan tyhjällä (white space); tehdään siksi sanojen etsintää varten oma funktio
- merkkien luku lopetetaan, kun kohdataan EOF- merkki (End Of File).

Ohjelma on seuraavanlainen lopullisessa muodossaan:

```
/* Esimerkkiohjelma sanojen laskemisesta  */
#include <stdio.h>
#include <ctype.h>

int found_next_word();

main()
{
  int word_cnt = 0;

  while (found_next_word() == 1){
    ++word_cnt;
  }
  printf("Sanojen lukumaara = %d\n", word_cnt);
}

int found_next_word()
{
  int c;
  while( isspace(c = getchar() )){
    ;                          /* ohita tyhja tila */
  }
```

```
  if ( c != EOF) {
    while ( (c = getchar() ) != EOF && !isspace(c)){
      ;                        /* ohita kaikki muu paitsi tyhja ja EOF */
    }
    return (1);                /* loydetty uusi sana */
  }
  return (0);                  /* uutta sanaa ei loydetty */
}
```

2.6 Aritmeettinen laskenta

2.6.1 Tyyppimuunnokset

Usein joudutaan tilanteeseen, jossa suure, joka halutaan sijoittaa muuttujan arvoksi ei olekaan oikeaa tyyppiä (esimerkiksi reaalilukumuuttujalle laskutoimituksen tuloksena sijoitettava arvo on kokonaislukutyyppinen). Toisaalta usein halutaan käyttää sekaisin esimerkiksi kokonaisluku- ja reaalilukutyyppisiä muuttujia tai yksinkertaisen ja kaksinkertaisen tarkkuuden reaalilukuja. C- kääntäjä suorittaa näissä tapauksissa automaattisesti tyypin muunnoksen.

Tyypinmuunnos (cast) voidaan suorittaa myös lausekkeella:

```
(t) e
```

missä `t` on tyyppimäärite (int, float, double ja jne.) ja `e` on jokin lauseke. Tyypinmuunnoslausekkeen arvo on `e`:n arvo muuttettuna `t`- tyyppiseksi. Tyypinmuunnos on tarpeen esimerkiksi, kun funktion kutsussa funktiolle on välitettävä määrättyä tyyppiä oleva parametri. Esimerkiksi, jos muuttujan `r` tyyppi olisi `float`, sen neliöjuuri lasketaan lausekkeella `sqrt( (double) r)`, koska funktion `sqrt` argumentin tulee olla tyyppiä `double`.

2.6.2 Matemaattiset funktiot

C- kielessä ei ole sisäänrakennettuja funktioita. Siten matemaattiset funktiot kuten potenssiinkorotus tai neliöjuuri on laskettava käyttäen hyväksi erityistä matemaattisten funktioiden kirjastoa. Kun matemaattisia funktioita käytetään, on `include`- direktiivilla otettava mukaan header- tiedosto `math.h`. Tärkeimmät matemaattiset funktiot ovat:

sqrt(x)	neliöjuuri ($\sqrt{x}$)
pow(x,y)	potenssiinkorotus (x^y)
exp(x)	eksponenttifunktio (e^x)
log(x)	logaritmifunktio
sin(x)	sinifunktio
cos(x)	kosinifunktio
tan(x)	tangenttifunktio
fabs(x)	itseisarvo
ceil(x)	lähinnä suurempi kok.luku
floor(x)	lähinnä pienempi kok.luku
fmod(x,y)	jakojäännös

Huomaa, että sekä **funktion arvo** että sen **parametrit** ovat tyyppiä `double`.

2.6.3 Esimerkki

Esimerkkinä yksinkertaisesta aritmeettisesta laskennasta tarkastellaan ohjelmaa, joka laskee korkoa korolle. Käytetään taas top- down- design menetelmää, joilloin tehtävä voidaan suorittaa seuraavilla funktioilla:

```
#include <stdio.h>

void prn_instructions();
void prn_results(double amount, double interest_rate,
        int nyears, double principal);
double compute(double interest_rate,
        int nyears, double principal);

main()
{
  int nyears;
  double amount,
         interest_rate,
         principal;

  prn_instructions();
  printf(" Anna paaoma, korko, vuosien lukumaara: ");
  scanf("%lf%lf%d", &principal, &interest_rate, &nyears);
  amount = compute(interest_rate, nyears, principal);
  prn_results(amount, interest_rate, nyears, principal);
}

void prn_instructions()
{
  printf("\n%s\n%s\n%s\n%s\n%s\n%s\n%s\n%s\n",
     "Tama ohjelma laskee korkoa korolle vuosittain, kun",
     "paaoma, korko, vuosien lukumaara on annettu.",
     "Esimerkiksi kun halutaan laskea korko",
     "1000 markan talletuksella,",
     "jonka korko on 4.5% ja talletusaika 15 vuotta",
     "on annettava:",
     "  1000.0  0.045  15");
}

void prn_results(double amount, double interest_rate,
        int nyears, double principal)
{
  printf("\n%s%g%c\n%s%d%s\n\n",
    "       Korko: ", 100.0 * interest_rate, "%",
    "Talletusaika: ", nyears, " vuotta");
  printf("%s%9.2f\n%s%9.2f\n%s%9.2f\n\n",
    " Paaoma jakson alussa: ", principal,
    "        Paaoman kasvu: ", amount -principal,
    "Paaoma jakson lopussa: ", amount);
}

double compute(double interest_rate,
        int nyears, double principal)
{
  int    i;
  double amount;
```

```
  amount = principal;
  for (i = 1; i <= nyears; ++i)
    amount *= 1.0 + interest_rate;
  return (amount);
}
```

Esimerkki

```
/*      esimerkki matemaattisista funktioista*/

#include <stdio.h>
#include <math.h>
main()
{
        int     i,
                j,
                k;
        double  x,
                y;
        i       = ceil(3.5);
        j       = ceil(-3.5);
        k       = floor(-3.5);
        x       = fmod(-3.5, 3.0);
        y       = fabs(-17.4);
        printf("i = %2d j = %2d k = %2d \n", i, j, k );
        printf("x = %5.2lf y = %5.2lf \n", x, y );
}
```

2.7 Pointterit (osoittimet)

Pointteri (osoitin) on C- kielessä muuttuja, jonka arvona on muistipaikan osoite.

Tarkastellaan aluksi kahta esimerkkiohjelmaa, jotka pyrkivät selventämään pointterien käyttöä ja tarkoitusta:

```
#include<stdio.h>
void swap(int a, int b);
main()
{
  int x=12, y=24;
  printf("x= %d  y= %d\n", x, y);
  swap(x,y);
  printf("x= %d  y= %d\n", x, y);
}

void swap(int a,int b)
{
  int temp;
  temp = a;
  a = b;
  b = temp;
}
```

Kun ohjelma suoritetaan saadaan tulokseksi:

```
x = 12  y = 24
x = 12  y = 24
```

Tämä ei ilmeisesti ollut ohjelmansuunnittelijan tarkoitus.

Tarkoitusksena oli ilmeisesti vaihtaa muuttujien `x` ja `y` arvot keskenään. Näin ei kuitenkaan tapahtunut, koska funktiolle `swap` välitetään C- kielessä vain parametrien arvot kutsuhetkellä ("call by value") eikä niiden osoitteita ("call by reference). Sen vuoksi muuttujilla `x` ja `y` on edelleen alkuperäiset arvonsa funktion `swap` suorituksen jälkeenkin.

Oikein toimiva ohjelma olisi seuraavanlainen:

```
#include<stdio.h>
void swap(int *a, int *b);
main()
{
  int x, y;
  x = 12;
  y = 24;
  printf("x= %d  y= %d\n", x, y);
  swap(&x, &y);
  printf("x= %d  y= %d\n", x, y);
}

void swap(int *a,int *b)
{
  int temp;
  temp = *a;
  *a = *b;
  *b = temp;
}
```

Kun nyt suoritetaan tämä korjattu ohjelma saadaan tulokseksi:

```
x = 12  y = 24
x = 24  y = 12
```

Funktiolle `swap` on välitetty muuttujien osoitteet. Tässä luvussa selvitetään, miten tämä tapahtuu.

2.7.1 Pointterien määrittely

Pointteri on muuttuja, jonka arvona on muistipaikan **osoite**. Tästä johtuu nimitys pointteri eli osoitin. Pointterit määritellään lisäämällä muuttujan nimen eteen $*$.

`Muoto:`

```
tyyppi *muuttuja;
```

Esimerkiksi

```
int *i;      /* viittaa kokonaisluvun osoitteeseen */
double *d;   /* viittaa double luvun osoitteeseen */
char *merkki;/* viittaa merkkitiedon osoitteeseen */
```

2.7.2 Osoitteet ja pointterit

Lisäämällä operaattori `&` muuttujan eteen saadaan muuttujan osoite; siis, jos `x` on muuttuja, niin `&x` on kyseisen muuttujan osoite koneen muistissa. Operaattoria $*$ puolestaan käytetään määrittelyn lisäksi myös operaattorin `&` käänteisoperaattorina, eli `*x` antaa sen muistipaikan sisällön, jonka osoite on muuttujan `x` arvo.

Seuraava ohjelma ehkä selventää pointterin arvon ja sen muistipaikan sisällön ("derefenced value") eroa:

```
main()
{
  int i = 11, *p;

  p = &i;
  printf(" i:n arvo   = %d\n", *p);
  printf(" i:n osoite = %d\n", p);
 }
```

Tulos voisi olla esimerkiksi (muistiosoitteen arvo riippuu käytetystä tietokonejärjestelmästä):

```
i:n arvo   =  11
i:n osoite = 2147479212
```

2.7.3 Pointterien alustaminen

Arvon antaminen (alustaminen) pointterimuuttujalle tapahtuu sijoittamalla sen arvoksi jonkin muistipaikan osoite `&` operaattoria käyttäen. Muistipaikan numeroarvoa ei ole syytä käyttää alustuksessa. Myös mitä tahansa pointterityyppistä lauseketta voidaan käyttää alustamiseen.

```
#include <stdio.h>
main()
{
        int     x,
                y,
                *p;
        x       = 5;
        p       = &x;
        y       = *p;
        printf(" x = %d ja y = %d\n",x,y);
}
```

Jos muuttuja on globaali, pitää lausekkeen olla kuitenkin vakioarvoinen. Globaalien muuttujien määritettely tapahtuu ohjelman alussa ennen main funktiota.

2.7.4 Pointterin kasvattaminen

Pointterin arvoa voidaan kasvattaa ja vähentää. Askeleena on tyyppimäärittelyn mukainen määrä tavuja.

```
int luku=5, *i;
double x=10.7, *d;
        i = &luku;
        i++;
        d = &x;
        d++;
```

2.7.5 Pointterit funktioiden argumentteina

Jos halutaan, että muutokset, jotka tehdään funktiossa muodollisille parametreille, välittyvät takaisin pääohjelmaan, niin silloin sekä todelliset että muodolliset parametrit pitää määritellä pointtereiksi. On huomattava, että usein, kun käytetään muodollisina parametreina pointtereita, myös funktion rungossa laskutoimitukset suoritetaan silloin pointterinmuuttujilla.

Esimerkkinä järjestetään kaksi lukua suuruusjärjestykseen:

```
#include <stdio.h>
void order(int *a,int *b);

main()
{
  int x1=24, x2=12;
  printf("x1= %d  x2= %d\n", x1, x2);
  order(&x1, &x2);
  printf("Suuruusjarjestyksessa:\n");
  printf("x1= %d  x2= %d\n", x1, x2);
}

void order(int *a,int *b)
{
  int temp;
  if (*a > *b) {
    temp = *a;
    *a = *b;
    *b = temp;
  }
}
```

2.8 Taulukot ja pointterit

2.8.1 Yksiulotteiset taulukot

Taulukko on olio, jota käyttäen yhdellä nimellä voidaan viitata moneen muistipaikkaan. Yksittäinen muistipaikka ilmoitetaan taulukon nimen perään liitetyllä kokonaisluvulla. C- kielessä taulukot määritellään lisäämällä muuttujan määrittelyn yhteydessä tunnuksen jälkeen hakasulkuihin numero, joka ilmoittaa, kuinka monta alkiota taulukossa on.

Muoto:

```
tyyppi taulukko[lukumäärä];
```

Esimerkiksi

```
int i[5];      /* viittaa kokonaislukutaulukkoon*/
double d[100];   /* viittaa double lukujen taulukkoon */
char merkki[15]; /* viittaa merkkien taulukkoon */
```

Ohjelmissa taulukon alkioihin viitataan indeksoiduilla muuttujilla, joissa indeksi kirjoitetaan hakasulkuihin. Yksidimensioinen taulukko vastaa vektoria matematiikassa. On kuitenkin huomattava, että

matematiikassa vektorin indeksit numeroidaan ykkösestä alkaen, kun taas C- kielen taulukoissa **indeksointi alkaa nollasta (0)**. Helpointa on silloin varata taulukko, jonka koko on yhtä alkiota suurempi kuin vektorin koko ja jättää taulukon ensimmäinen (eli siis alkio $x[0]$) käyttämättä.

Esimerkkinä taulukoiden käytöstä lasketaan, kuinka monta kertaa kukin iso kirjain esiintyy syötetyssä tekstissä:

```
#include <stdio.h>
#include <ctype.h>

main()
{
  int c, i, letter[26];

  for (i=0; i < 26; ++i){
    letter[i] = 0;
  }
  while ( (c=getchar() ) != EOF){
    if( isupper(c)){
      ++letter[c-'A'];
    }
  }
  for(i=0; i<26; ++i) {
    if(i % 6 == 0) {
      printf("\n");
    }
    printf("%5c:%4d", 'A'+i, letter[i]);
  }
  printf("\n");
}
```

Huomaa, että koska esimerkissä käytetään merkkifunktioita, niin on otettava mukaan otsikkotiedosto `ctype.h`.

2.8.2 Taulukoiden alustus

Taulukkomuuttuja alustetaan kirjoittamalla määrittelyn yhteydessä taulukon arvot aaltosulkuihin yhtäläisyysmerkin jälkeen. Jos listassa on vähemmän arvoja kuin taulukossa on alkioita, ylimääräisten alkioiden alkuarvoksi asetaan nolla. Siten taulukko voidaan nollata kayttämällä alkuarvoa [0].

Esimerkkejä taulukoiden alustuksesta:

```
float a[100] = {0.0};
int month[12] = {31, 28, 31, 30, 31, 30,
                 31, 31, 30, 31, 30, 31}
```

Ensimmäinen esimerkki nollaa kaikki taulukon `a` alkiot alkiosta `a[0]` alkioon `a[99]`.

2.8.3 Taulukoiden käsittely pointterien avulla

C- kielessä taulukon nimi viittaa taulukon ensimmäisen alkion osoiteeseen tietokoneen muistissa. Taulukon seuraavat alkiot on sijoitettu muistissa seuraaviin osoitteisiin. Siten taulukon nimeä voidaan käyttää kuten pointteria, joka viittaa ensimmäiseen alkioon. Lisäämällä pointteriin kokonaisluku n voidaan viitata taulukon n:nteen alkioon.

Siis, kun lausekkeessa esiintyy taulukon nimi ilman indeksointia, se tarkoittaa pointteria taulukon ensimmäiseen alkioon; `p = a` tarkoittaa samaa kuin `p = &a`[0].

Toisaalta, jos osoittimen nimen perään pannaan hakasulkeissa numero, niin osoitin käyttäytyy, kuten taulukko, jonka ensimmäinen alkio (indeksi=0) on nimen ilmoittamassa osoitteessa. Esimerkiksi olkoon p määritelty osoittimeksi `p`[5] ja `*(p+5)` viittaavat saman muistipaikan sisältöön (ei ositteeseen).

2.8.4 Pointteriaritmetiikka ja taulukon indeksit

Seuraavassa esimerkissä ryhmitellyt rivit tarkoittavat samaa.

```
double taulu[10], *pd1, *pd2, luku1, luku2, luku3;

pd1 = taulu;
pd2 = &taulu[0];

pd1 = pd1 + 1;
pd2 = &taulu[1];

luku1 = *(pd1+1);
luku2 = pd1[1];
luku3 = taulu[2];
```

2.8.5 Esimerkkejä

Esimerkki 1:

```
/* suurin.c, suurin luku */
#include <stdio.h>
main()
{
        int     taulu[100], i=0, raja, j, testi=-1;
        printf(" anna posit. kok.lukuja, viimeisenä -1\n");
        scanf ("%d",&taulu[i]);
        while ( taulu[i]!= -1){
                i++;
                scanf ("%d",&taulu[i]);
        }
        raja    = --i;
        for ( j=0;j<=raja;j++ ) {
                if(taulu[j]>testi) {
                        testi   = taulu[j];
                }
        }
        printf(" suurin luku on %3d\n",testi);
}
```

Esimerkki 2:

```
/* lotto.c, lottorivin takastus */
#include <stdio.h>
main()
{
        int taulu[40]={0}, luku, oma, i=0, j, oikein=0;
        printf(" anna oikea lottorivi\n");
        for(i=1;i<=7;i++) {
```

```
                scanf ("%d",&luku);
                taulu[luku]     = 1;
        }
        printf(" anna oma rivisi\n");
        for(i=1;i<=7;i++) {
                scanf ("%d",&oma);
                if (taulu[oma]== 1) {
                        oikein++;
                }
        }
        printf(" rivissäsi oli %2d oikein\n",oikein);
        printf(" oikea rivi on \n");
        for ( j=1;j<=39;j++ ) {
                if(taulu[j] == 1) {
                        printf("%4d\n",j);
                }
        }
}
```

2.8.6 Taulukot funktion argumentteina

Funktion argumentti voidaan määritellä taulukoksi. Määrittelyssä voi kuitenkin tässä tapauksessa jättää taulukon koon tyhjäksi. Funktiolle välitetään itseasiassa tällöin vain taulukon ensimmäisen alkion osoite eli taulukko käyttäytyy kuten pointteri.

Esimerkkinä lasketaan kahden vektorin skalaaritulo eli summa $\sum_{i=1}^{n} x_i \cdot y_i$:

```
/* pistetulo */
#include<stdio.h>
#define N 100

void read_vec(float x[], int n);
float dot_prod(float x[], float y[], int n);

main()
{
  float x[N], y[N];
  int n;

  printf("Vektorin alkioiden lukumaara: ");
  scanf("%d", &n);
  printf("anna x vektorin komponentit ");
  read_vec(x, n);
  printf("anna y vektorin komponentit ");
  read_vec(y, n);
  printf("Skalaaritulo = %7.2e\n", dot_prod(x, y, n));
}

void read_vec(float x[], int n)
{
  while( (n--) > 0) {
    scanf("%f", &x[n]);
  }
```

```
}

float dot_prod(float x[], float y[], int n)
{
  float  sum = 0.0;
  while( (n--) > 0){
    sum += x[n]  * y[n];
  }
  return (sum);
}
```

Sama esimerkki pointterien avulla esitettynä:

```
#include<stdio.h>
#define N 100

void read_vec(float *x, int n);
float dot_prod(float *x,float *y, int n);

main()
{
  float x[N], y[N];
  int n;

  printf("Vektorin alkioiden lukumaara: ");
  scanf("%d", &n);
  printf("anna x vektorin komponentit ");
  read_vec(x, n);
  printf("anna y vektorin komponentit ");
  read_vec(y, n);
  printf("Skalaaritulo = %7.2e\n", dot_prod(x, y, n));
}

void read_vec(float *x, int n)
{
  while( (n--) > 0){
    scanf("%f", x++);
  }
}

float dot_prod(float *x,float *y, int n)
{
  float  sum = 0.0;
  while( (n--) > 0){
    sum += *(x++) * *(y++);
  }
  return (sum);
}
```

2.8.7 Moniulotteiset taulukot

Moniulotteiset taulukot määritellään C- kielessä taulukkoina, joiden alkiot ovat taulukoja. Siten voidaan määritellä esimerkiksi taulukot:

```
float matriisi[10][10];
int taulu[2][4][10];
```

Hakasuluissa olevat numerot ilmoittavat kunkin indeksin mahdollisten arvojen lukumäärän; indeksien alarajana on jälleen nolla (0).

Koska moniulotteinen taulukko on yksiulotteinen taulukko, jonka alkioina on taulukoita, se sijaitsee muistissa vaakariveittäin.

Moniulotteisen taulukon ollessa funktion muodollisena parametrina sen määrittelyssä vain *ensimmäisen* indeksin arvojen lukumäärä saa puuttua.

Esimerkki Kahden alkion vaihto 2-ulotteisessa taulukossa.

```
#include <stdio.h>
int swap(int taulu[][3],int i1,int i2, int i3);

main()
{
        int luvut[2][3]={{7,2,9},
                         {4,6,8}};
        int rivi = 1, sara1 = 1, sara2 = 3, i, j;
        swap(luvut,rivi-1,sara1-1,sara2-1);
        printf("taulukko\n");
        for(i=0;i<=1;i++) {
                for(j=0;j<=2;j++) {
                        printf("%d  ",luvut[i][j]);
                }
                printf("\n");
        }
}

int swap(int taulu[][3],int i1,int i2, int i3)
{
        int     apu;
        apu             = taulu[i1][i2];
        taulu[i1][i2]   = taulu[i1][i3];
        taulu[i1][i3]   = apu;
}
```

2.8.8 Tiedon välitys funktioiden välillä

Tähän mennessä olemme käsitelleet tiedon välitystä funktioiden välillä seuraavilla eri tavoilla

1. Globaali muuttuja
2. Todellisten parametrien arvon sijoitus muodollisten parametrien arvoksi
3. Tulosparametri
4. Saman muistipaikan käyttö pointterien tai taulukoiden avulla

Esimerkkinä summan laskeminen funktiossa eri tiedonvälitymekanismeja käyttäen

```
/*      globaalien muuttujien käyttö tiedon välitykseen
        aliohjelman ja kutsuvan ohjelman välillä
*/
#include <stdio.h>

int     summa();
int     a, b;

main()
{
        int     c;
        a       = 1;
        b       = 1;
        printf("a = %d b = %d\n",a,b);
        c       = summa();
        printf(" %d + %d = %d\n",a,b,c);
}

int summa()
{
        a       = 2;
        b       = 2;
        return(a+b);
}



/*      tulosparametrin käyttö tiedon välitykseen
        aliohjelmasta kutsuvaan ohjelmaan
*/
#include <stdio.h>
int summa(int x,int y);

main()
{
        int     a = 1, b = 1, c;
        printf("a = %d b = %d\n",a,b);
        c       = summa(a,b);
        printf(" %d + %d = %d\n",a,b,c);
}

int summa(int x,int y)
{
        return(x + y);
}



/*      tulosparametrin käyttö tiedon välitykseen
        aliohjelmasta kutsuvaan ohjelmaan
        Huom! parametrien muutettuja arvoja ei saa
        palautettua sijoituksella
*/
```

```
#include <stdio.h>

int summa(int x, int y);

main()
{
        int     a=1,  b=1 , c;
        printf("a = %d b = %d\n",a,b);
        c       = summa(a,b);
        printf(" %d + %d = %d\n",a,b,c);
}

int summa(int x, int y);
{
        x       = 2;
        y       = 2;
        return(x + y);
}



/*      osoittimien käyttö tiedon välitykseen
        aliohjelmasta kutsuvaan ohjelmaan
*/
#include <stdio.h>

int summa(int *x, int*y);

main()
{
        int     a=1, b=1, c;
        printf("a = %d b = %d\n",a,b);
        c       = summa(&a,&b);
        printf(" %d + %d = %d\n",a,b,c);
}

int summa(int *x, int *y)
{
        *x      = 2;
        *y      = 2;
        return(*x + *y);
}
```

2.9 Merkkijonot ja pointterit

C- kielessä merkkijonot esitetään yksidimensioisina taulukkoina, joiden tyyppi on `char`. Yksittäisiä merkkejä voidaan siten käsitellä joko taulukon alkioina tai pointterien avulla.

Määrittely:

```
char merkkijono[pituus+1];
```

Standardikirjastossa on monia merkkijonojen käsittelyyn tehtyjä funktioita.

2.9.1 Merkkijonon loppumerkki

Kun merkkijono tallennetaan taulukkoon, taulukon viimeiseen alkioon tallennetaan merkkijonon loppumerkiksi merkki \0. Sen vuoksi merkkijonolle on varattava taulukko, jonka koko on **ainakin yhden alkion** verran **suurempi** kuin merkkijonon pituus.

Kun taulukkoon `nimi`

```
char nimi[5];
```

tallennetaan merkkijono `Aapo`, niin taulukon alkioiden sisältö on seuraava

```
nimi[0] = 'A';
nimi[1] = 'a';
nimi[2] = 'p';
nimi[3] = 'o';
nimi[4] = '\0';
```

eli taulukon `nimi` pituus pitää olla tässä tapauksessa vähintään viisi alkiota.

Esimerkki merkkijonon määrittelystä ja käytöstä:

```
/*  Esimerkkiohjelma hello
*/
#include <stdio.h>
#define PITUUS 100

main()
{
  char nimi[PITUUS];

  printf("Mika sinun nimesi on: ");
  scanf("%s", nimi);

  printf("\nHello %s , have a nice day!\n", nimi);
}
```

Kun ohjelma suoritetaan, funktio `scanf` lukee **yhden sanan** (eli merkkijonon, jonka jälkeen tulee tyhjää: joko blanko, tabulaattori tai rivinvaihto). Sanan edessä mahdollisesti olevat tyhjät merkit ohitetaan.

Merkkijonoja käytettäessä on huomattava, että `scanf`- funktiossa **ei käytetä** muuttujan edessä &-merkkiä. Syynä on se, että merkkijonomuuttuja on taulukko ja sen pelkkä nimi pointteri, joka viittaa taulukon ensimmäiseen muistipaikkaan. Merkintä `nimi` on silloin sama kuin, jos `scanf`- funtion parametrina olisi käytetty lauseketta `&nimi[0]`.

2.9.2 Merkkijonojen alustus

Kuten muutkin taulukot myös merkkijonot voidaan alustaa. Tämä voidaan tehdä kahdella eri tavalla;

```
char nimi[] = {'A', 'a', 'p', 'o', '\0'};
```

kuten muissakin taulukoissa tai käyttämällä merkkijonovakiota alustukseen

```
char nimi[] = "Aapo";
```

On huomattava, että ensimmäisessä tavassa myös loppumerkki on lisättävä taulukon viimeiseksi alkioksi, mutta jälkimmäisesä tavassa se tehdään automaattisesti. Lisäksi on huomattava, että merkkijonon pituuden määrää nyt automaattisesti alustettavan merkkijonon pituus eli yllä olevassa esimerkissä merkkijonotaulukkon `nimi` pituudeksi tulee viisi alkiota.

Kolmas tapa on käyttää pointteria merkkijonon määrittelyyn ja alustamiseen

```
char *nimi = "Aapo";
```

2.9.3 Merkkijonojen käsittely

Koska merkkijono on merkkejä sisältävä taulukko, sitä voidaan käsitellä taulukon indeksien avulla.

Esimerkkinä kirjoitetaan ohjelma, joka lukee yhden rivin tekstiä ja tulostaa sen jälkeen rivin käänteisessä järjestyksessä:

```
/* Esimerkkiohjelma palindrom.c */
#include <stdio.h>
#define MAXLINE 80

main()
{
  char c, line[MAXLINE];
  int i;

  printf("\nKirjoita rivi tekstia: \n");
  for (i=0; (c = getchar()) != '\n'; i++) {
    line[i] = c;
  }
  line[i] = '\0';

  printf("\nRivi on\n");
  printf("%s%s\n", "etuperin:  ", line);
  printf("%s",     "takaperin: ");
  while (i>=0) {
    putchar(line[i--]);
  }
  printf("\n");
}
```

Koska merkkijono on taulukko, sitä ei voi käyttää sijoituslauseissa, vaan sijoitus on tehtävä käyttämällä funktiota `strcpy`.

Esimerkiksi kopioidaan merkkijono toiseen merkkijonoon ja verrataan ovatko ne tosiaan samat:

```
#include <stdio.h>
#include <string.h>
main()
{
  char mj1[10] = {'a', 'b', 'c', '\0'};
  char mj2[10];

  strcpy(mj2, mj1);
  printf("%s\n%s\n", mj1, mj2);
  printf("vertailu: %d\n", strcmp(mj1,mj2));
```

```
  mj2[3] = 'd';
  mj2[4] = '\0';
  printf("%s\n%s\n", mj1, mj2);
  printf("vertailu: %d\n", strcmp(mj1,mj2));

}
```

2.9.4 Merkkipointterit

Taulukon indeksien sijasta merkkijonojen käsittelyssä voidaan käyttää merkkipointtereita.

Kun merkkijono esiintyy funktion argumenttina, se tarkoittaa pointteria (osoitinta) merkkijonon alkuun (taulukon alkioon 0). Pointteria voidaan käyttää myös merkkijonojen käsittelyyn, kuten yksittäisten merkkien poimimiseen jonosta.

Esimerkkinä esitetään funktio, joka kopioi merkkijonon toiseen merkkijonomuuttujaan sekä taulukon indeksointia että pointteriaritmetiikkaa käyttämällä.

Taulukoiden avulla tehty funktio:

```
/* kopiointi taulukoiden avulla */
mkopio(char x[],char y[])
{
  int i;
  for (i=0; x[i] != '\0'; i++) {
    y[i] = x[i];
  }
  y[i] = '\0';
}

/* kopiointi pointterien avulla */
mkopio(char *x,char *y)
{
  while ((*y++ = *x++) != '\0') {
    ;
  }
}
```

2.9.5 Merkkijonopointteritaulukot

C- kielessä voidaan määritellä taulukkoja mille tahansa C- kielen muuttajatyypille ja siten myös merkkipointterien taulukoita eli taulukoita jonka alkiot ovat pointterita merkkijonoihin.

Esimerkiksi:

```
  char *day[] = {"Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"};
  char *paiva[] = {"Su", "Ma", "Ti", "Ke", "To", "Pe", "La"};
```

2.10 main-funktio

Ohjelmassa on yksi ja vain yksi main-funktio. Jos main-funktion tyyppiä ei ole ilmoitettu, niin se on oletusarvon mukaisesti tyyppiä `int`. Huolellinen ohjelmoija käyttää tyyppimääritystä `void` myös `main`-funktiolle silloin, kun se ei palauta mitään arvoa.

Funktiolla `main` on kaksi argumenttia,

```
main(int argc, char *argv[]),
```

joista ensimmäinen (`argc`) ilmoittaa argumettien lukumäärän ja toinen argumentti (`argv`) on pointteri taulukkoon, joka sisältää itse argumetit merkkijonoina. Ensimmäisenä argumenttina (`argv`[0]) pidetään ohjelman nimeä.

Suoritettavan C- kielisen ohjelman yleinen muoto on silloin

```
ohjelma arg1 arg2 arg3  < syote > tuloste
```

eli ohjelmalle voidaan antaa mielivaltainen määrä (yllä olevassa esimerkissä kolme kappaletta) argumentteja ja syöttö sekä tulostus voidaan uudelleenohjata, kuten millä tahansa DOS- komennolla.

Esimerkkinä tehdään ohjelma, joka tulostaa argumenttinsa näytölle:

```
/*      main argumenttien tulostus näytölle
*/
#include <stdio.h>

void main(int argc, char *argv[])
{
  int i;

  printf("Argumenttien lukumaara, argc = %d\n", argc);
  printf("Ohjelman nimi on, argv[0]    = %s\n", argv[0]);
  printf("Argumentit ovat:\n");
  for(i=1; i<argc; i++)
  printf("                  argv[%d]    = %s\n", i, argv[i]);

}
```

Ohjelman tulostus on:

```
C> main toimiiko tama ohjelma
Argumenttien lukumaara, argc = 4
Ohjelman nimi on, argv[0]    = a.out
Arguumentit ovat:
                  argv[1]    = toimiiko
                  argv[2]    = tama
                  argv[3]    = ohjelma
```

2.11 Header tiedostot, direktiivit ja preprosessori

C-kielen rakenteesta johtuen ohjelma tarvisee aina jonkin tai joitakin header tiedostoja; mm syöttön ja tulostuksen suorittavat funktiot on sijoitettu header tiedostoihin. `Include`-direktiiviä käyttäen saadaan header tiedostot eli valmiit funktiot mukaan ohjelmaan.

2.11.1 `#include`-direktiivi

`include`-direktiivi voi olla joko muotoa

```
#include <tiedosto.h>
```

tai muotoa

```
#include "tiedosto.h"
```

Direktiivin vaikutuksesta annetun tiedoston sisältö sijoitetaan mukaan käännökseen kyseisen rivin kohdalle. Jälkimmäisessä tapauksessa tutkitaan ensin onko `tiedosto.h` senhetkisessä työhakemistossa, ja jos se ei ole, niin sitä etsitään määrätyistä systeemihakemistoista (Borland C++- kääntäjä etsii tavallisesti hakemistosta `\borlandc\include`). Edellisessä tapauksessa tiedostoa ei yritetä etsiä ollenkaan työhakemistosta vaan ainoastaan systeemihakemistoista.

Borlandin C++ sisältää mm seuraavat header tiedostot. Help-valikosta on löydettävissä hyvät selitykset kaikkien funktioiden toiminnoista esimerkkeineen. Tähän monisteeseen on kerätty vain muutamia esimerkkejä.

2.11.2 stdio.h

Funktiot:

```
clearerr    fclose      fcloseall
fdopen      feof        ferror
fflush      fgetc       fgetchar
fgetpos     fgets       fileno
flushall    fopen       fprintf
fputc       fputchar    fputs
fread       freopen     fscanf
fseek       fsetpos     ftell
fwrite      getc        getchar
gets        getw        perror
printf      putc        putchar
puts        putw        remove
rename      rewind      scanf
setbuf      setvbuf     sprintf
sscanf      strerror    _strerror
tmpfile     tmpnam      ungetc
unlink      vfprintf    vfscanf
vprintf     vscanf      vsprintf    vsscanf
```

Esimerkki getchar-funktio

```
{\bf getchar} Macro that gets character from stdin

 Syntax:
   int getchar(void);

 Prototype in:
 stdio.h

On success, getchar returns the character read, after converting it to
an int without sign extension. On end-of-file or error, it returns EOF.

 #include <stdio.h>

 int main(void)
 {
    int c;

    /* Note that getchar reads from stdin and
```

```
       is line buffered; this means it will
       not return until you press ENTER. */

    while ((c = getchar()) != '\n')
       printf("%c", c);

    return 0;
 }
```

Esimerkki scanf-funktio

```
{\bf scanf} Performs formatted input from stdin

Syntax:
   int scanf(const char *format [, ...]);

Prototype in:
 stdio.h

Returns the number of input fields processed successfully. It processes
input according to the format and places the results in the memory
locations pointed to by the arguments.
```

2.11.3 math.h

```
Funktiot:

abs       acos      asin      atan
atan2     atof      cabs      ceil
cos       cosh      exp       fabs
floor     fmod      frexp     hypot
labs      ldexp     log       log10
matherr   modf      poly      pow
pow10     sin       sinh      sqrt
tan       tanh
```

Esimerkki abs- funktio

```
{\bf abs} Macro that gets the absolute value of an integer

Syntax:
  real:     int abs(int x);
  complex:  double abs(complex x);

Prototype in:
math.h    stdlib.h (real)    complex.h (complex)

#include <stdio.h>
#include <math.h>

int main(void)
```

```
{
  int number = -1234;

  printf("number: %d  absolute value: %d\n", number, abs(number));
  return 0;
}
```

2.11.4 string.h

Funktiot:

```
memccpy     memchr      memcmp
memcpy      memicmp     memmove
memset      movedata    movmem
setmem      stpcpy      strcat
strchr      strcmp      strcmpi
strcpy      strcspn     strdup
_strerror   strerror    stricmp
strlen      strlwr      strncat
strncmp     strncmpi    strncpy
strnicmp    strnset     strpbrk
strrchr     strrev      strset
strspn      strstr      strtok      strupr
```

2.11.5 conio.h

```
cgets           clreol
clrscr          cprintf
cputs           cscanf
delline         getch
getche          getpass
gettext         gettextinfo
gotoxy          highvideo
insline         kbhit
lowvideo        movetext
normvideo       putch
puttext         _setcursortype
textattr        textbackground
textcolor       textmode
ungetch         wherex
wherey          window
```

2.11.6 `#define` direktiivi ja makrot

`define`- direktiiviä käytetään kahdessa eri muodossa. Symbolisia vakioita määritellään lauseella, joka on muotoa

```
#define tunnus arvo
```

ja makroja lauseella, joka on muotoa

```
#define tunnus(argumenttilista) teksti
```

Symbolisten vakioiden tunnukset kirjoitetaan tavallisesti isoilla kirjaimilla; esimerkiksi

```
#define PI 3.141
#define STRINGSIZE  100
```

Huomaa, että `define`- direktiivin yhteydessä **ei käytetä** yhtäsuuruusmerkkiä eikä lausetta lopeteta puolipisteellä.

Makroja käytettäessä tunnuksen jälkeen ei saa jättää tyhjää vaan vasemman sulun tulee olla heti tunnuksen jälkeen, muutoin **preprosessori** ei osaa käsitellä makroa oikein. Makroja käytetään usein lyhyiden funktioiden asemasta. Makrot eroavat funktioista siinä, ettei niiden argumenteilla ole tyyppiä.

Esimerkiksi funktio `min`:

```
/*
 * min
 * ***
 * Returns mimimum of the two integers (a or b).
 */
  int min(a,b)
  int a,b;
  {
    return( a<b ? a : b);
   }
```

voitaisiin korvata makrolla:

```
#define min(a,b) ( ((a)<(b)) ? (a) : (b) )
```

Huomaa, että sulkuja on käytetty runsaasti varmistaamaan, että operaatiot suoritetaan oikeassa järjestyksessä.

2.11.7 Ehdollinen kääntäminen

Preprosessori tuntee kontrollikomentoja, joilla voidaan valita käännetäänkö määrätyt rivit vai ei. Kontrollilausekkeet ovat:

```
#if vakiolauseke
#ifdef tunnus
#ifndef tunnus
#else
#endif
```

Jos `if`- lausekkeessa vakiolauseke on nollasta eroava (tosi) sen jälkeen tulevat rivit suoritetaan seuraavaan `endif`- lausekkeella alkavaan riviin saakka. Vastaasti `ifdef`- lausekkeella alkava koodi suoritetaan, jos tunnus on määritelty `define`- direktiivillä.

Usein näitä kontrollirakenteita käytettään, kun halutaan kirjoittaa koodia, jonka pitää toimia useissa eri järjestelmissä.

Esimerkiksi ohjelmassa voisi olla rivit

```
#ifdef(AIX)

#include <sys/time.h>
#include <sys/resource.h>

#else
```

```
#include <time.h>
#include <resource.h>

#endif
```

jolloin ensimmäinen osa käännetäisiin silloin, kun käyteään AIX- käyttöjärjestelmää ja toinen osa kun käytetään esimerkiksi Turbo C- kääntäjää MS- DOS järjestelmässä.

Vastaavasti voidaan valita esimerkiksi jotkut tulostuskäskyt suoritettaviksi vain määrätyissä tapauksissa. **Esimerkiksi** voitaisiin määritellä symbolinen vakio `DEBUG` otsikkotiedostossa

```
/* otsikkotiedosto valitse.h */
#define DEBUG 1
```

ja sisällyttää se ohjelmatiedostoon `include`- direktiivillä

```
/* varsinainen ohjelmatiedosto */
#include "valitse.h"

#if DEBUG
  printf(" debug: testaus alkaa\n")
  ....
#endif
```

Kun ohjelma on testattu ja se toimii ylimääräiset tulostuskäskyt voidaan ohittaa muuttamalla symbolisen vakion `DEBUG` arvoksi nolla (0) otsikkotiedostossa `valitse.h` ja kääntämällä ohjelma sen jäleen uudelleen.

2.12 Syöttö ja tulostus

C- kielessä syöttö ja tulostus kuvataan laiteriippumattomasti tietovirralla (data stream). Tietovirta voi kuvata joko näppäimistöä ja näyttöä tai levytiedostoja. Tietovirta voi olla

- tekstivirta: jono merkkejä, jotka muodostavat rivejä ja kunkin rivin lopussa on rivinvaihtomerkki
- binäärivirta: jono tavuja, jotka sisältävät tietoa koneen sisäisessä esitysmuodossa

Tietovirta liitetään ulkoiseen tiedostoon (tai laitteeseen) avaamalla tiedosto. Tiedoston ja tietovirran välinen yhteys taas poistetaan sulkemalla tiedosto.

C- ohjelma avaa aina käynnistyessään automaattisesti kolme tekstivirtaa (tiedostoa):

- standardisyötövirta (standard input): `stdin`
- standarditulostusvirta (standard output): `stdout`
- standardivirhevirta (standard error): `stderr`

Tavallisesti `stdin` on näppäimistö ja `stdout` sekä `stderr` on näyttö, jolloin sekä tulostus että virheilmoitukset tulostetaan sekaisin näytölle.

Funktiolla `scanf` luetaan standardisyöttövirtaa ja funktiolla `printf` tulostetaan standarditulostusvirtaan.

2.12.1 Syöttö- ja tulostustiedostot

Standarditietovirrat ovat aina avoinna, mutta muut tiedostot on avattava ennenkuin niitä voidaan käyttää ja suljettava sen jälkeen, kun niiden käyttö lopetetaan.

Tiedosto avataan funktiolla `fopen`

```
Syntax:

       FILE *fopen(const char *filename, const char *mode);
```

jonka tyyppi on `FILE`.

Funktio palauttaa pointterin avattuun tietovirtaan, mikäli avaus on onnistunut, muuten palautuu arvo `NULL`. Tietovirtaa kontrolloiva muuttuja on tyyppiä `FILE` oleva pointteri.

```
Käyttö:
FILE *tietovirta
.
.
tietovirta = fopen("tiedosto.dat","r");
```

Tässä levyllä oleva tiedosto "`tiedosto.dat`" avataan. Siihen viitataan ohjelman sisällä — aina kun tarve vaatii — pointterilla `tietovirta`.

Funktion `fopen` ensimmäinen argumentti on siis levyllä oleva tai sinne luotava tiedosto.

Funktion `fopen` toinen argumentti määrää moodin:

- "r"avaa vanhan tiedoston lukemista varten; avaaminen epäonnistuu jos tiedostoa ei ole olemassa
- "w"luo uuden tiedoston kirjoittamista varten. Jos samalla nimellä esiintyy tiedosto, niin se hävitetään.
- "a"avaa (vanhan tai uuden) tiedoston lisäyksiä varten. Uusi teksti tulee entisen loppuun.
- "+"lisäysmerkki (w+), joka sallii tiedoston päivityksen.
- "t"kirjoitus/luku tekstimuodossa.
- "b"kirjoitus/luku binaarimuodossa.

Kun ohjelman suoritus lopetetaan, niin kaikki tietovirrat suljetaan. Jos tiedosto halutaan sulkea ennen ohjelman suorituksen päättymistä, se voidaan tehdä funktiokutsulla `fclose`.

```
 Syntax:
   int fclose(FILE *stream);
```

Funktio fclose palauttaa arvon 0, jos sulkeminen on onnistunut, muuten EOF-merkin.

Esimerkkinä hahmotellaan komento, joka lukee datansa tiedostosta, jonka nimi annetaan komennon parametrina tai siinä tapauksessa, että komennolle ei anneta yhtään parametria standardisyötöstä (näppäimistöltä):

```
#include <stdio.h>

main(argc, argv)
int argc;
char *argv[];
{
  FILE *tietovirta;
```

```
  .....

  if(argc == 1)
    tietovirta = stdin;
  else
    tietovirta = fopen(argv[1], "r");

  .....

  fclose(tietovirta)
}
```

2.12.2 Muotoiltu tulostus: funktiot `printf, fprintf`

Funktiolla `fprintf` tulostus kirjoitetaan tulostusvirtaan muotoiltuna:

```
fprintf(tietovirta, muotoilu, arg1, ..., argn);
```

missä muotoilu määrää tulostuksen ulkoasun ja `arg1`,...,`argn` ovat tulostettavia lausekkeita. `fprintf`-funktio on evkivalentti funktion `printf` kanssa paitsi, että käyttäjä voi siinä määritellä tulostustiedoston `tietovirta` kun taas `printf` tulostaa aina standarditulosvirtaan `stdout`.

Esimerkkinä ohjelma, jolla suoritetaan tiedoston AUTOEXEC.BAT varmuustallennus.

```
#include <stdio.h>

int main(void)
{
   FILE *in, *out;

   if ((in = fopen("\\AUTOEXEC.BAT", "rt"))
       == NULL)
   {
      fprintf(stderr,
              "Cannot open input file.\n");
      return 1;
   }

   if ((out = fopen("\\AUTOEXEC.BAK", "wt"))
       == NULL)
   {
      fprintf(stderr,
              "Cannot open output file.\n");
      return 1;
   }

   while (!feof(in))
      fputc(fgetc(in), out);

   fclose(in);
   fclose(out);
   return 0;
}
```

2.12.3 Formatoitu syöttö: funktiot `scanf, fscanf`

Funktiolla `fscanf` luetaan muotoiltua syöttöä syöttövirrasta:

```
fscanf(tietovirta, muotoilu, arg1, ..., argn);
```

missä muotoilu määrää syöttötietojen tyypit ja `arg1`,...,`argn` ovat luettavien muuttujien tunnukset. Funktio `fscanf` on ekvivalentti funktion `scanf` kanssa paitsi, että käyttäjä voi siinä määritellä syöttötiedoston `tietovirta` kun `scanf` lukee aina standardisyötöstä `stdin`.

2.12.4 Muotoilu

Syntax:

```
% [leveys] [.tarkkuus] [kokomäärite] tyyppi
```

Esim. %d, %10.3f %15.5lf

Muotoilun ohjaus alkaa aina (%) merkillä ja päättyy muotoilutyyppiin. Kaikki merkit, jotka ovat muotoilumerkkijonossa muotoilunohjausten ulkopuolella, tulostuvat sellaisinaan.

Tyyppi:

tyyppi	tulostuksen muotoilu
d	kokonaisluku int
i	kokonaisluku int
o	unsigned kahdeksan järjestelmän luku int
u	unsigned kokonaisluku int
x	In printf = unsigned hexadecimal int; in scanf = hexadecimal int
X	In printf = unsigned hexadecimal int; in scanf = hexadecimal long
f	Liukuluku [-]dddd.ddd
e	Liukuluku [-]d.ddd e [+/-]ddd
g	Format e or f valinta takkuuden perusteella
E	Sama kuin edellä, mutta E eksponentin eteen
G	Sama kuin edellä, mutta E eksponentin eteen
c	yksi merkki
s	merkkijono merkkiin '\0' tai tarkkuuden asettamaan rajaan
saakka	
%	merkki %
p	pointteri

Tarkkuus: Reaalilukuja tulostettaessa tarkkuus kertoo desimaalipisteen jälkeen tulostettavien desimaalien lukumäärän. Merkkijonoa tulostettaessa takkuus kertoo tulostettavien merkkien enimmäismäärän.

.tarkkuus	tulostus
(tyhjä)	oletusarvo
.0 (d,i,o,u,x)	oletusarvo
(e,E,f)	ei desimaaleja
.n	enintään n merkkiä

Leveys on positiivinen luku, joka kertoo tulostettavan kentän vähimmäisleveyden. Kentässä mahdollisesti vajaaksi jäävään kohtaan tulostetaan välilyöntejä.

```
  leveys tulostus

    n       vähintään n merkkiä, tyhjät lisätään vesemmalle
   0n       vähintään n merkkiä, tyhjät lisätään oikealle
```

Kokomäärite: Merkit h, l ja L

```
    h     d,i,o,u,x,X    argumentti on short int
    l     d,i,o,u,x,X    argumentti on long int
    l     e,E,f,g,G      argumentti on double (scanf only)
    L     e,E,f,g,G      argumentti on long double
```

2.12.5 Virheilmoitukset

Funktiolla `perror` voidaan tulostaa virheilmoituksia tietovirtaan.

Esimerkiksi edellisessä esimerkissä voitaisiin testata onnistuuko tiedoston avaaminen lukemista varten:

```
#include <stdio.h>
main(argc, argv)
int argc;
char *argv[];
{
  FILE *tietovirta;

  .....

  if(argc == 1)
    tietovirta = stdin;
  else
    if ((tietovirta = fopen(argv[1], "r")) == NULL)
      {
perror("Syottotiedoston avaus epaonnistui");
exit(1)
      }

  .....

  fclose(tietovirta)
}
```

2.12.6 Esimerkkejä

Esimerkki 1, jossa nimi, numero pareja lisätään tiedostoon.

```
#include <string.h>
#include <stdio.h>

int main(void)
```

```
{
  FILE *tiedosto;
  char nimi[6], numero[6], c;
  int arvo;
  tiedosto = fopen("puhlu.dat","a");
  printf("lisätäänkö nimi ja numero y/n\n");
  while((c=getchar())!='n') {
   printf("anna nimi\n");
   scanf("%s",nimi);
   printf("anna numero\n");
   scanf("%s",numero);
   fflush(stdin);
   printf("%s %s\n",nimi,numero);
   fprintf(tiedosto,"%5s %5s\n",nimi,numero);
   printf("lisätäänkö nimi y/n\n");
  }
}
```

Esimerkki 2, jossa luetaan ja tulostetaan tiedoston `puhlu.dat` luettelon sisältö. Tiedosto sisältää nimi, numero pareja.

```
#include <string.h>
#include <stdio.h>

int main(void)
{
  FILE *tiedosto;
  char nimi[6], numero[6];
  int i;
  tiedosto = fopen("puhlu.dat","rt");
  while(fscanf(tiedosto,"%5s %5s\n",nimi,numero)!=EOF) {
        printf("%s %s\n",nimi,numero);
  }
}
```

Esimerkki, jossa korjataan `puhlu.dat` luettelon sisältöä. Tiedosto sisältää nimi, numero pareja.

```
#include <string.h>
#include <stdio.h>

int main(void) {
    FILE *tiedosto;
    char nimi[6], numero[6] , c;
    int i;
    tiedosto = fopen("puhlu.dat","r+");
    while(fscanf(tiedosto,"%5s %5s\n"
                    ,nimi,numero)!=EOF){
      printf("%s %s\n",nimi,numero);
      printf("korjataanko y/n\n");
      if((c=getchar())=='y') {
         fseek(tiedosto,-12,SEEK_CUR);
         printf("anna uusi nimi\n");
         scanf("%s",nimi);
```

```
         printf("anna uusi numero\n");
         scanf("%s",numero);
         fprintf(tiedosto,"%5s %5s\n",nimi,numero);
         fseek(tiedosto,0,SEEK_CUR);
       }
       fflush(stdin);
    }
}

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
\begin{verbatim}
    SYNOPSIS
          #include <stdio.h>

          int fseek (FILE *stream, long offset, int ptrname);

          void rewind (FILE *stream);

          long ftell (FILE *stream);

     DESCRIPTION
          fseek sets the position of the next input or output
          operation on the stream [see intro(3)].  The new position is
          at the signed distance offset bytes from the beginning, from
          the current position, or from the end of the file, according
          to a ptrname value of SEEK_SET, SEEK_CUR, or SEEK_END
          (defined in stdio.h) as follows:

          SEEK_SET    set position equal to offset bytes.

          SEEK_CUR    set position to current location plus offset.

          SEEK_END    set position to EOF plus offset.
```

2.13 Tietorakenteet

Tietue (structure) on joukko loogisesti yhteenkuuluvia tietoja, jotka usein ovat kuitenkin keskenään erityyppisiä.

2.13.1 Tietueen määrittely

Tietueen (structure) määrittely on tyypillisesti muotoa:

```
struct htieto {
  char *etunimi;
  char *sukunimi;
  char *stunnus;
  int  vkoe1;
  int  vkoe2;
  int  arvosana;
};
```

Tässä on määritelty tietue, jonka kenttinä (field) ovat henkilötiedot ja välikokeiden pisteet sekä loppuarvosana. **Huomaa**, että oikeanpuoleisen aaltosulun jälkeen tulee puolipiste. Tämän jälkeen tyyppiä `struct htieto` olevia muuttujia voitaisiin määritellä seuraavalla tavalla:

```
struct htieto kalle, ville;
struct htieto opiskelijat[100];
```

Taulukkoon `opiskelijat` mahtuisi siten sadan opiskelijan suoritusmerkinnät.

Muuttujat voidaan määritellä myös samalla kertaa itse tietueen määrittelyn kanssa kirjoittamalla muuttujien tunnukset oikeanpuoleisen aaltosulun jälkeen

```
struct htieto {
  char *etunimi;
  char *sukunimi;
  char *stunnus;
  int  vkoe1;
  int  vkoe2;
  int  arvosana;
} anna, bertta;
```

2.13.2 Tietueen alustus

Tietue alustetaan kirjoittamalla yhtäsuuruusmerkin jälkeen aaltosulkuihin tietueen kenttien arvot pilkulla toisistaan erotettuna. Jos arvoja on vähemmän kuin kenttiä, ylimääräiset kentät saavat arvon nolla.

Esimerkiksi määritellään ensin tietue paivamaara

```
struct paivamaara {
  int paiva;
  int kuukausi;
  int vuosi;
};
```

ja sen jälkeen kaksi muuttujaa ja samalla alustetaan ne

```
struct paivamaara joulu = { 24, 12, 1991};
struct paivamaara vappu = { 1 , 5 , 1992};
```

Tällaista rakennetta voi käyttää **vain** muuttujien alustuksessa.

Vastaavasti tietuetaulukko alustetaan seuraavasti:

```
struct paivamaara htyot[3] = { { 20, 11, 1991},
       { 27, 11, 1991},
       {  4, 12, 1991} };
```

2.13.3 Kenttämuuttuja

Tietuemuuttujan `x` kenttään `a` viitataan kirjoittamalla piste tietueen ja kentän nimien väliin.

```
x.a
```

Kenttämuuttuja voidaan käyttää kuten samantyyppistä tavallista muuttujaa. **Esimerkiksi** edellä määritelty vappupäivä voidaan tulostaa lauseella:

```
printf(" Seuraava vappu on %d paivana %d kuuta %d vuonna %s\n",
       vappu.paiva, vappu.kuukausi, vappu.vuosi);
```

Esimerkkinä syötetään välikoetulokset tietueisiin `htieto` ja tulostetaan ne ruudulle.

```
#include <stdio.h>

int main(void)
{
    int i, luku;
    struct htieto {
      char etunimi[20];
      int  vkoe1;
      int  vkoe2;
    };
    struct htieto opiskelijat[100];
    printf("lukumäärä\n");
    scanf("%d",&luku);
    printf("anna etunimi vkoe1 vkoe2\n");
    for(i=1;i<=luku;i++){
       scanf("%5s,%d,%d",opiskelijat[i].etunimi
                        ,&opiskelijat[i].vkoe1
                        ,&opiskelijat[i].vkoe2);
    }
    printf("\n");
    for(i=1;i<=luku;i++){
       printf("%5s,%3d,%3d\n"
                  ,opiskelijat[i].etunimi
                  ,opiskelijat[i].vkoe1
                  ,opiskelijat[i].vkoe2);
    }
}
```

2.13.4 Tietueet ja pointterit

Tietuiden käsittelyssä käytetään usein pointtereita, jolloin sitä varten C- kielessä on määritelty lyhennysmerkintä

`p -> a`

rakenteelle

`(*p).a`

kun `p` on pointterityyppinen lauseke, joka osoittaa tietueesen, missä on `a`- niminen kenttä.

2.13.5 Tietueet ja funktiot

Tietueiden välittäminen funktion argumentteina sekä palauttaminen funktion arvona on mahdollista.

Esimerkiksi C- kielessä ei ole määritelty kompleksilukuja, mutta voimme itse helposti kirjoittaa tarvittavat funktiot ja määrittelyt. Määritellään kahden reaaliluvun muodostama pari tietueeksi, joka edustaa kompleksilukua ja sen jälkeen funktiot kompleksilukujen yhteen- ja kertolaskua varten:

```
struct complex {
  floar r;                  /* reaaliosa */
  float i;                  /* imaginaariosa */
}

struct complex csumma(struct complex a, struct complex b)
```

```
{
  struct complex c;
  c.r = a.r + b.r;
  c.i = a.i + b.i;
  return (c);
}

struct complex ctulo(struct complex a, struct complex b)
{
  struct complex c;
  c.r = a.r * b.r  -  a.i * b.i;
  c.i = a.i * b.r  +  a.r * b.i;
  return (c);
}
```

Näitä funktioita käytettäisiin seuraavasti:

```
#include <stdio.h>

struct complex{
  float r;                  /* reaaliosa */
  float i;                  /* imaginaariosa */
};

struct complex csumma(struct complex a, struct complex b);
struct complex ctulo(struct complex a, struct complex b);

void main()
{
  struct complex z1 = { 1.0, 2.0};
  struct complex z2 = { 1.5, 2.5};
  struct complex z3;

  z3 = csumma(z1, z2);
  printf(" Summa = %f + %f i \n", z3.r, z3.i);

  z3 = ctulo(z1, z2);
  printf(" Tulo  = %f + %f i \n",  z3.r, z3.i);
}
```

Liite A

ASCII- koodi

luku	ASCII-merkki
31	
32	
33	!
34	"
35	#
36	$
37	%
38	&
39	’
40	(
41	)
42	*
43	+
44	,
45	-
46	.
47	/
48	0
49	1
50	2
51	3
52	4
53	5
54	6
55	7
56	8
57	9
58	:
59	;
60	<
61	=
62	>
63	?
64	@
65	A
66	B

67	C
68	D
69	E
70	F
71	G
72	H
73	I
74	J
75	K
76	L
77	M
78	N
79	O
80	P
81	Q
82	R
83	S
84	T
85	U
86	V
87	W
88	X
89	Y
90	Z
91	[
92	\
93	]
94	^
95	_
96	‘
97	a
98	b
99	c
100	d
101	e
102	f
103	g
104	h
105	i
106	j
107	k
108	l
109	m
110	n
111	o
112	p
113	q
114	r
115	s
116	t
117	u
118	v

```
119   w
120   x
121   y
122   z
123   {
124   |
125   }
126   ~
127
```

Liite B

Operaattorien sidontajärjestys

Liite C

Vanhoja tenttikysymyksiä

1. Kirjoita ohjelma, joka lukee näppäimistöltä toistuvasti lukuja niin kauan kunnes luettu luku on nolla, jonka jälkeen ohjelma tulostaa ruutuun luettujen lukujen lukumäärän sekä suurimman luetun luvun.

2. Laadi seuraava ohjelma jollakin kielellä. Ohjelman tulee piirtää kuvaruutuun N kpl samankeskisiä ympyröitä. N luetaan näppäimistöltä samoin kuin piirrettävien ympyröiden suurin ja pienin säteen arvo sekä keskipisteen koordinaatit (x, y). Oleta, että ympyrän piirto tapahtuu käskyllä $circle(x, y, r)$ jossa r on ympyrän säde, ja että kuvaruutu saadaan ns. grafiikkatilaan käskyllä $screen(1)$.

3. Muodosta tietokanta, joka sisältää laitoksen sisäisen puhelinluettelon muodossa

NIMI	NUMERO
PALO JARI	341
OJA ARI	123
SALO MARI	456
...	...

   Tee sen jälkeen ohjelma, jolla tulostetaan päätteeltä annetun henkilön puhelinnumero.

4. Mitä tarkoitetaan ohjelmointikielessä toistorakenteella? Laadi esimerkkinä ohjelma, joka lukee näppäimistöltä toistuvasti kokonaislukuja yksi kerrallaan, kunnes luettu luku = 0, jonka jälkeen ohjelma tulostaa suurimman luetun luvun.

5. Tee ohjelma, joka tulostaa annetut neljä nimeä aakkosjärjestyksessä.

6. Mitä tulostuu seuraavassa ohjelmassa?

```
#include <stdio.h>
main()
{
       int     taulu[3][3]={{1,2,3},
                            {4,5,6},
                            {7,8,9}},
               i,
               j;
        for ( i=2;i>=0;i-- ) {
             for( j=0;j<=2;j++) {
                  printf(" %3d  ",taulu[j][i]);
```

```
            }
            printf("\n");
        }
}
```

7. Mitä tarkoitetaan ohjelmointikielessä valintarakenteella? Esimerkkinä laadi ohjelma, joka tulostaa annetut kolme lukua suuruusjärjestyksessä.

8. Mitä tulostuu seuraavassa ohjelmassa?

```
#include <stdio.h>
main()
{
        char merkki = 'c';
        int i, j, ind=0;
        for (i=1;i<=6;i++) {
                if(i<=3) {
                  ind++;
                }
                else {
                  ind--;
                }
                for (j=0;j<=ind;j++) {
                   printf( "%c",merkki);
                }
                printf("\n");
        }
}
```

9. Esitä kolme erilaista tapaa siirtää tietoa pääohjelman ja aliohjelmien eli funktioiden välillä C-kielessä.

10. Mitä tulostuu seuraavassa ohjelmassa?

```
#include <stdio.h>
int summa();
main()
{
       int a=6, b=8, c=10;
       c       = summa(&a,&b);
       printf(" %d + %d = %d\n",a,b,c);
}
int summa(x,y)
int     *x, *y;
{
        *x      = 3;
        *y      = 5;
        return(*x + *y);
}
```

11. Esittele jokin toistorakenne. Tee esimerkkinä ohjelma, joka lukee 5000 kpl lukuja päätteeltä ja tulostaa ne käänteisessä järjestyksessä.

12. Mitä tulostuu seuraavassa ohjelmassa?

```
#include <stdio.h>
main()
{
        char merkki = 'g';
        int i, j, ind=4;
        for (i=1;i<=7;i++) {
               if(i<=4) {
                    ind--;
               }
               else {
                    ind++;
               }
               for (j=0;j<=ind;j++) {
                     printf( "%c",merkki);
               }
               printf("\n");
        }
}
```

13. Selosta lyhyesti esimerkkien avulla, mitä ohjelmointikielessä tarkoitetaan muuttujalla, taulukolla ja aliohjelmalla.

14. Mitä tulostuu seuraavassa ohjelmassa?

```
#include <stdio.h>
main()
{
        char merkki = 'A';
        int i, j, ind=3;
        for (i=1;i<=6;i++) {
                if(i<=3) {
                  ind--;
                }
                else {
                  ind++;
                }
                for (j=0;j<=ind;j++) {
                   printf( "%c",merkki);
                }
                printf("\n");
        }
}
```

15. Tee ohjelma, joka lukee päätteeltä sanoja, viimeisenä sanan `LOPPU`, tutkii sanojen pituudet ja tulostaa pisimmän sanan viisi ensimmäistä kirjainta. Ohjelmointikielen voit itse valita.

16. Laadi ohjelma, joka lukee toistuvasti lukuja niin kauan kunnes syötetään 0. Ohjelman on tutkittava syötettyjä lukuja ja laskettava niiden lukujen lukumäärä, jotka olivat suurempia kuin 5.0, sekä erikseen negatiivisten lukujen lukumäärä. Edelleen ohjelman on ilmoitettava mikä oli suurin syötetty luku ja laskettava annettujen lukujen keskiarvo. Ohjelmointikielen voit itse valita, mutta goto-hyppykäskyn käyttäminen ei ole sallittua. Miten saat ohjelman toimimaan niin, että se tulostaa lopuksi kaikki luetut luvut käänteisessä järjestyksessä, so. viimeksi luettu luku ensin, jne.?

17. Laadi ohjelma, joka pyytää käyttäjältä kokonaisluvun *lkm* ja sitten lukee *lkm* kappaletta lukuja yksitellen näppäimistöltä. Kun viimeinen luku on syötetty ohjelma lajittelee ne ja tulostaa suuruusjärjestyksessä. (Vihje: Vertaa peräkkäisiä lukuja ja vaihda ne tarvittaessa keskenään toistuvasti niin kauan kuin on tarpeen.)

18. Laadi ohjelma, joka lukee päätteeltä luvun N ja N kpl lukuja taulukkoon $A(i), i = 1, ..., N$ sekä laskee ja tulostaa päätteelle ko. lukujen neliöiden summan neliöjuuren, ts. N-ulotteisen vektorin A pituuden. Lsäksi ohjelman on tulostettava taulukon A suurimman alkion arvo.

19. Laadi algoritmi ja kirjoita ohjelma, joka lukee tiedostosta SIVU.TXT korkeintaan 50 kpl tekstirivejä keskusmuistiin, järjestää ne rivien pituuden mukaan kasvavaan järjestykseen ja lopuksi tulostaa rivit ruutuun pituusjärjestyksessä. Merkkitietomuuttujan x$ pituus saadaan funktiolla LEN(x$).

20. Ratkaise jompikumpi seuraavista:

    (a) Laadi ohjelma, joka lukee tiedostosta NIMET.DAT korkeintaan 50 nimeä ja kirjoittaa ne aakkosjärjestyksessä tiedostoon ABC.DAT

    (b) Laadi ohjelma, joka etsii funktion nollakohtaa välillä A $\leq$ x $\leq$ B tarkkuudella EPS käänteistä lineaarista interpolointia käyttäen. A, B ja EPS luetaan näppäimistöltä. Olkoon tutkittava funktio f(x) = $xe^x - 3$

21. Laadi ohjelma, joka lukee toistuvasti lukuja niin kauan, kunnes syötetään luku 0. Ohjelman on laskettava syötettyjen negatiivisten lukujen lukumäärä sekä niiden lukujen lukumäärä, jotka olivat suurempia kuin 5.0. Edelleen ohjelman on tulostettava lukujen keskiarvo ja suurin syötetty luku.

22. Tee ohjelma, joka lukee päätteeltä sanoja, viimeisenä sanan `LOPPU`, tutkii sanojen pituudet ja tulostaa pisimmän sanan viisi ensimmäistä kirjainta.

23. Tee ohjelma, joka lukee N kpl positiivisia kokonaislukuja välillä 1...50 ja tutkii, mikä niistä esiintyy useimmin ja mikä vähiten. Ohjelma tulostaa nämä luvut ja niiden esiintymiskertojen lukumäärän. Jos useimmin ja vähiten esiintyviä lukuja on monta, niin ne tulostetaan suuruusjärjestysessä.

24. Laadi ohjelma, joka ensiksi lukee näppäimistöltä keskusmuistiin lukuja, kunnes luettu luku on nolla tai lukujen lukumäärä on 100. Toisessa vaiheessa ohjelman tulee etsiä luvuista suurin ja laskea niiden keskiarvo. Lopuksi ohjelman on tulostettava suurin luku, keskiarvo sekä kaikki luetut luvut käänteisessä järjestyksessä so. viimeksi luettu ensimmäiseksi jne. Ohjelmointikielen voit valita itse.

25. Laadi algoritmi ja kirjoita ohjelma, joka etsii aakkosjärjestykseen lajitellusta tiedostosta näppäimistöltä luetun nimen ja tulostaa sen järjestysnumeron kuvaruutuun. Jos nimeä ei löydy, tulostetaan ko. nimi ja teksti "Ei löydy". Oleta, että tiedostossa on korkeintaan 200 nimeä, jotka voidaan lukea kerralla keskusmuistiin.

26. Suorita lottoarvonta käyttäen satunnaislukugeneraattoria random(n), joka antaa satunnaisesti lukuja väliltä 0 < luku < n−1. Arvo siis 7 lukua väliltä 1 - 39, ja tulosta nämä luvut.

27. Tee ohjelma, joka järjestää N kpl lukuja suuruusjärjestykseen. Miten merkkitiedon järjestäminen aakkosjärjestykseen poikkeaa lukujen järjestelemisestä?

28. Esitä esimerkin avulla, kuinka taulukkoa käytetään tietojen välityksessä pääohjelman ja funktion välillä.

29. Laadi ohjelma, joka ensiksi lukee näppäimistöltä keskusmuistiin lukuja, kunnes luettu luku on nolla tai lukujen lukumäärä on 100. Toisessa vaiheessa ohjelman tulee etsiä luvuista suurin ja laskea niiden keskiarvo. Lopuksi ohjelman on tulostettava suurin luku, keskiarvo sekä kaikki luetut luvut suuruusjärjestyksessä (suurin luettu luku ensimmäiseksi jne).

30. Suorita lottoarvonta käyttäen satunnaislukugeneraattoria random(n)

    ```
    #include <stdlib.h>

    int random(int n)
    ```

    joka antaa satunnaisesti lukuja väliltä 0 < luku < n−1. Arvo siis 7 lukua väliltä 1 - 39, ja tulosta nämä luvut suuruusjärjestyksessä. Kun halutaan satunnaislukujen olevan satunnaisia eri ajokerroilla, on ohjelman alkuun lisättävä `randomize();`.

31. Esitä kaksi eri tapaa, joilla voidaan suorittaa toisto *Basic*-kielessä käyttäen esimerkkinä summan

$$\sum_{i=1}^{100} \frac{1}{i^2} = 1 + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \cdots + \frac{1}{100^2}$$

    arvon laskemista.

32. Tee ohjelma, joka lukee N-kpl kokonaislukuja välillä 1...1000 ja tulostaa ne suuruusjärjestyksessä. Tulostuksessa kukin luku esiintyy vain kerran, vaikka syötössä sallitaankin samojen lukujen antaminen.

33. Esitä algoritmi, jolla annettua nimeä voidaan hakea aakkosjärjestykseen lajitellusta nimiluettelosta. Kirjoita myös ohjelma jollakin kielellä.

34. Esitä jokin algoritmi lukujen järjestämiseksi suuruusjärjestykseen. Miten merkkitiedon järjestäminen aakkosjärjestykseen poikkeaa lukujen järjestelemisestä ? Kirjoita algoritmi ohjelmaksi jollakin kielellä.

35. Tee ohjelma, joka piirtää x,y-koordinaatiston kuvaruudulle ja tähän koordinaatistoon suoran

$$y = ax + b$$

    kuvaajan. Kertoimet a ja b luetaan näppäimistöltä.

36. Esittele lyhyesti seuraavat käsitteet

    (a) käyttöjärjestelmä
    (b) tiedosto
    (c) valinta C-kielessä
    (d) open-lause
    (e) komentotiedosto (tyyppiä nimi.BAT)
    (f) tulostuksen ohjaus käyttöjärjestelmän alaisuudessa.

37. Laadi algoritmi, jonka avulla aakkosjärjestykseen lajitellusta luettelosta etsitään annettua nimeä. Laadi myös ohjelma, joka etsii aakkosjärjestykseen lajitellusta tiedostosta NIMET.ABC annettua nimeä ja tulostaa kuvaruutuun nimen sijaintikohdan. Jos nimeä ei löydy, tulostetaan sijaintikohdan arvona nolla. Voit olettaa, että luettelo mahtuu kokonaisena keskusmuistiin.

38. Selvitä lyhyesti ns. tiedonhallintaohjelmiston käyttötarkoitusta ja toimintamuotoja. Miten menettelet jos tehtäväksi annetaan opiskelijoita koskevan suoritusrekisterin laatiminen. Rekisteriin tulisi sisällyttää kustakin opiskelijasta ainakin seuraavat tiedot: Nimi, osoite, koulutusohjelma, aloitusvuosi ja suoritetut kurssit arvosanoineen. Oleta, että koulutusohjelmassa on mahdollista suorittaa ainakin 50 eri kurssia.

39. Mitä etua on aliohjelmien käytöstä ohjelmoinnissa? Esimerkkinä laadi aliohjelma, jossa lasketaan annettujen lukujen keskiarvo. Käytä sitä sitten pääohjelmassa päätteeltä syötettyjen lukujen keskiarvon laskemiseen.

40. Laadi aliohjelma eli funktio, joka laskee taulukkoon **int luvut**[**100**] sijoitettujen kokonaislukujen keskiarvon. Kirjoita myös pääohjelma, jossa luvut luetaan näppäimistöltä ja laskettu keskiarvo tulostetaan ruudulle.