Johdatusta ohjelmointiin

Tämä tiedosto täydentää Saarelan ja Puoskarin luentomonistetta.

Mistä ohjelmoinnissa on kyse?

- Ohjelmointi on käskyjen antamista koneelle
- Onnistunut ohjemointisuoritus edellyttää, että ohjelmoija
  - osaa jakaa monimutkaisen tehtävän riittävän yksinkertaisiksi käskyiksi, jotka kone osaa suorittaa (algoritmin muodostaminen)
  - osaa esittää nämä käskyt kielellä, jota kone ymmärtää (ohjelmointikielen hallinta)
- Tietokone käsittelee tietoa viime kädessä ykkösinä ja nollina (bitteinä)
- Ohjelmointikielessä tietyn toimenpiteen aikaan saavat konekieliset käskyt on niputettu ohjelmointikielisiksi käskyiksi
- *Kääntäjä* on ohjelma, joka kääntää tietyllä ohjelmointikielellä kirjoitetut käskyt konekielelle
- Kääntäjä luo ohjelmointikielellä kirjoitetuista käskyistä suoritettavan tietokoneohjelman

Esimerkki 1

Lasketaan kahden positiivisen kokonaisluvun jakolaskusta jäävä jakojäännös koneella, jonka matemaattiset taidot rajoittuvat lukujen yhteen- ja vähennyslaskuun ja suuruuden vertailuun. Merkitään jaettavaa n:llä ja jakajaa k:lla. Oletetaan, että $n \geq k$.

1. Laske $n - k$.
2. Vertaa saatua lukua k:hon. Jos se on suurempi tai yhtä suuri kuin k, vähennä siitä edelleen k ja palaa tämän kohdan alkuun. Jos se on pienempi kuin k, lopeta algoritmi. Saatu luku on kysytty jakojäännös.

Tietokoneen muisti

- Jotta tietokone voisi käsitellä esim. lukuja tai tekstiä (yleisemmin *muuttujia*), ne on kirjoitettava sen muistiin
- Muisti on periaatteessa pitkä bittijono, joka edelleen ryhmitetään tavuihin (8 bittiä), kilotavuihin (2^{10} tavua), jne.
- Kun muuttuja määritellään (sille määrätään nimi ja tyyppi), sille varataan tilaa muistista

- Muuttujan tyyppi määrää, kuinka paljon muistitilaa se vie (riippuu ohjelmointikielestä)
- Lukuja on helppo esittää bittien avulla (binäärijärjestelmä), kirjoitusmerkkien esittämiseen on olemassa ASCII-koodi, jossa jokainen merkki vastaa tiettyä 8 bitin jonoa eli tavua
- Muuttujan nimi kertoo koneelle muistipaikan (muuttujan osoitteen tietokoneen muistissa), josta kone hakee muuttujan arvon (muistipaikan sisällön)
- Muistipaikan sisältöjä muutetaan sijoituslauseilla, jonka muoto lähes kaikissa ohjelmointikielissä on

```
muuttuja = lauseke
```

- Yhtäsuuruusmerkki = ei tarkoita yhtälöä sanan matemaattisessa mielessä. Merkin vasemmalla puolella oleva muuttuja saa arvokseen oikealla puolella olevan lausekkeen arvon.

Esimerkki 2

Edellinen algoritmiesimerkki voidaan kirjoittaa jo tietokoneohjelmaa muistuttavaan muotoon

1. Merkitse $p = n - k$.
2. Jos $p \geq k$, merkitse $p = p - k$ ja palaa tämän kohdan alkuun. Jos $p < k$, lopeta algoritmi.

Tämän algoritmin (ohjelmanpätkän) suorituksen jälkeen muuttujalla p on arvo, joka vastaa kysyttyä jakojäännöstä.

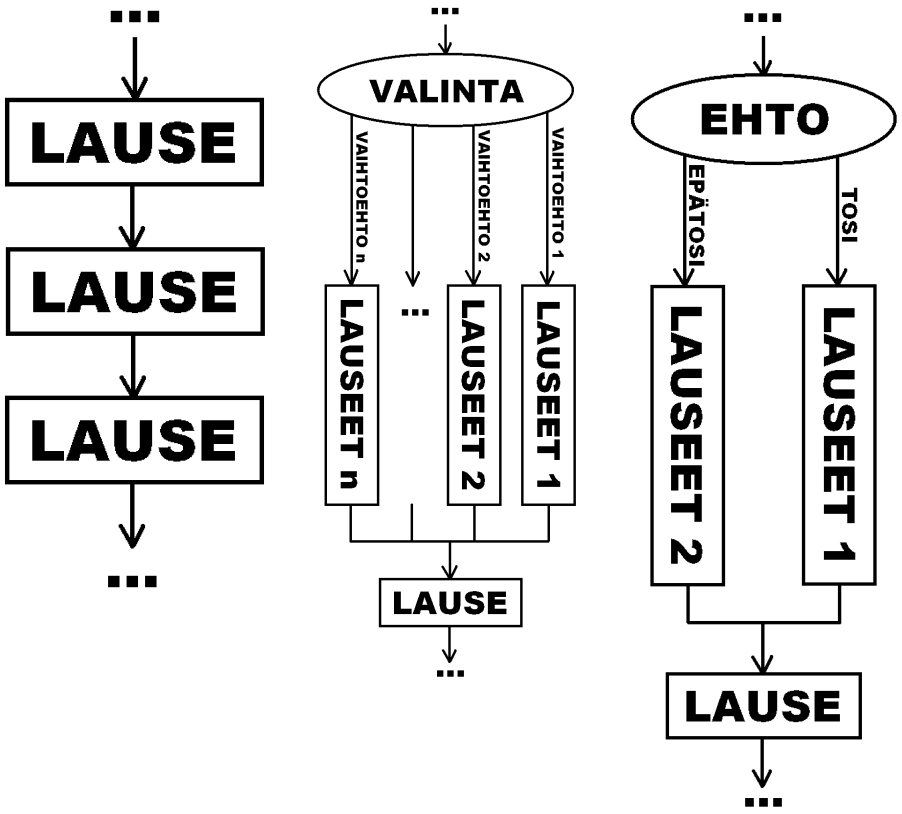
Ohjelman ohjausrakenteita

- Tavallisimmin ohjelmaan kirjoitetut käskyt suoritetaan kirjoitusjärjestyksessä, kukin yhden kerran
- Suoritusjärjestykseen voi vaikuttaa esim. seuraavasti
  - suoritettava lause valitaan annetuista vaihtoehdoista, riippuen jonkin ehtolausekkeen arvosta (valintarakenne)
  - jotkut lauseet suoritetaan toistuvasti, kunnes jokin ehto täyttyy (toistorakenne)
- Esimerkki valintarakenteen käytöstä on valikko, ts.

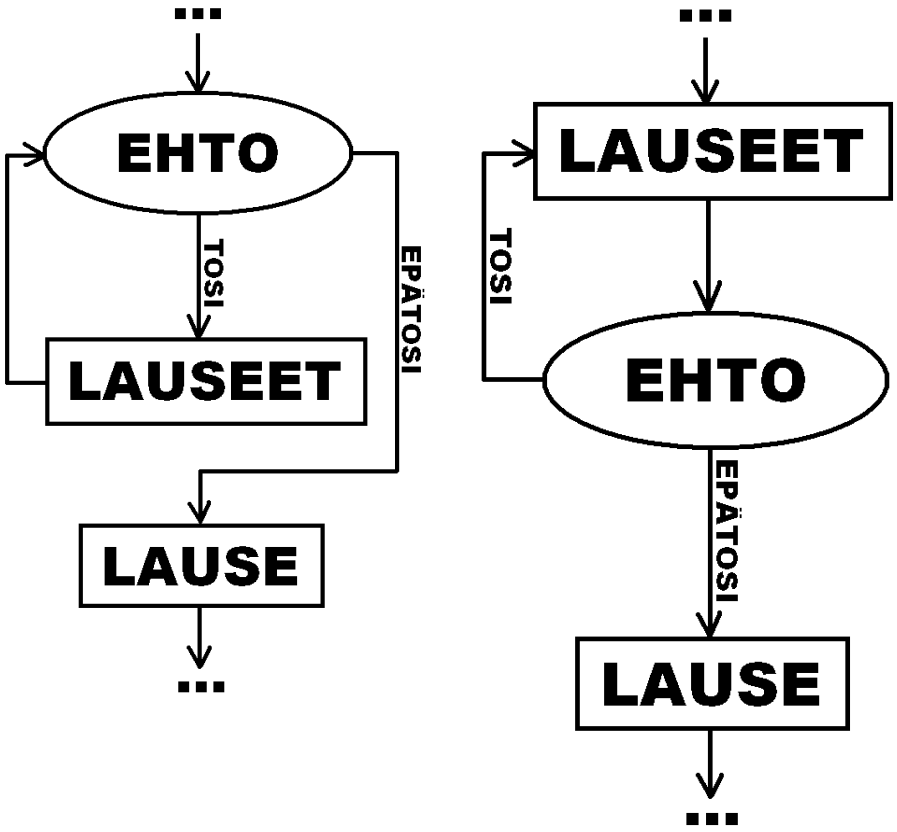
```
Valitse toiminto.

1. Luo uusi tiedosto
2. Avaa tiedosto
3. Syötä tietoja
4. Muokkaa tietoja
5. Lopeta ohjelma
```

- Ohjelman suorituksen eteneminen riippuu siitä, minkä vaihtoehdon käyttäjä valitsee
- Tärkeä erikoistapaus valintarakenteesta on valinta kahdesta vaihtoehdosta (tosi/epätosi, kyllä/ei,kruuna/klaava)
  - Tämä vastaa suomen kielen "Jos ..., tehdään ..., muuten tehdään ..." -rakennetta (vrt. jakojäännösalgoritmi)
  - Melkein kaikissa ohjelmointikielissä tämä valinta toteutetaan `if`-lauseella
- Ehto voi olla esim. lukujen vertailua. Yleisemmin ehtolauseke on jokin kokonaisluvun palauttava lauseke. Luku nolla tulkitaan vastaukseksi "epätosi" ja ykkönen vastaa vastausta "tosi".
- Ehtolauseita voi yhdistellä ns. loogisten konnektiivien (loogisten operaattorien) avulla, esim. `ehto1` **ja** `ehto2`, `ehto1` **tai** `ehto2`.
- Toistorakenteessa
  1. tarkastetaan jonkin ehdon paikkaansapitävyys
  2. suoritetaan lohko lauseita (toistettavat lauseet)
  3. tarkastetaan uudelleen ehdon paikkansapitävyys
  4. jne...
- Yleensä jokin toistettavista lauseista muuttaa jonkin ehtolausekkeessa esityvän muuttujan arvoa tms.
  - muutenhan ehdon totuusarvo ei koskaan muuttuisi!!!
- Jos halutaan toistaa jotkut lauseet esim. `n` kertaa, on luotava erillinen (kokonaislukutyyppinen) muuttuja, esim. `i`, jolla pidetään kirjaa suoritusten määrästä. Muuttujan `i` arvoa kasvatetaan yhdellä joka kierroksella ja jokaisen toiston jälkeen tarkistetaan lausekkeen `i<=n` totuusarvo.



Kuva 1: Valintarakenteet. Huom. useamman vaihtoehdon väliltä valitseminen voidaan toteuttaa sisäkkäisillä kahden vaihtoehdon väliltä valitsemisilla, mutta nämä menetelmät soveltuvat hiukan eri tarkoituksiin. Sisäkkäisillä 'jos'-lauseilla voidaan tarkastella useiden loogisten lausekkeiden totuusarvoja, kun taas useamman vaihtoehdon väliltä valittaessa suoritettavat lauseet määrää yhden ehtolausekkeen saamat arvot.



Kuva 2: Toistorakenteet. Huom. tässä esiintyvissä rakenteissa ei ole kovin syvällistä eroa – ne voidaan muuttaa toisikseen, esim. kirjoittamalla suoritettavat lauseet kertaalleen ennen toistorakenteen alkua.

Aliohjelmat

- Usein tarvittavia komentoryppäitä voi niputtaa uusiksi komennoiksi
- C-kielessä näitä kutsutaan *funktioiksi* tai *aliohjelmiksi*
- Funktiolle määritellään
  - Nimi
  - argumentit (lukumäärä ja kunkin argumentin tyyppi)
  - palautusarvo, jonka funktio *palauttaa* sitä kutsuvalle ohjelmalle (esim. aliohjelma palauttaa pääohjelmalle)
- Esimerkiksi yo. jakojäännöksen lasku on funktio, joka saa argumenteikseen kaksi kokonaislukua ja palauttaa kokonaisluvun (jakojäännöksen)
- On suositeltavaa jakaa ohjelmointitehtävä pieniin osiin ja määritellä kutakin osaa varten aliohjelma, joka suorittaa sen. Aliohjelmat voidaan edelleen jakaa osiin (huom. aliohjelmissa voidaan kutsua muita aliohjelmia, tai jopa itseään (rekursio))
- C-kielessä (ja monessa muussa) funktio on aina määriteltävä (ainakin esiteltävä) ennen ensimmäistä funktiokutsua, ts. ennen pääohjelman alkua. C-kielessä pätevät seuraavat säännöt:
  - Jos funktion määrittely on samassa tiedostossa kuin pääohjelma, funktion runko voidaan kirjoittaa esittelyn yhteyteen tai pääohjelman rungon jälkeen
  - Funktion määrittely voidaan kirjoittaa erilliseen tiedostoon (oltava tyyppiä `.h`, ts. kirjastotiedosto)

Esimerkki 3

Tehdään ohjelma, joka testaa, onko annettu positiivinen kokonaisluku n alkuluku. Käytetään hyväksi yo. funktiota, joka laskee jakojäännöksen. Luku n on alkuluku, jos jakojäännös jaettaessa n millä tahansa n:ää pienemmällä ja ykköstä suuremmalla kokonaisluvulla on erisuuri kuin nolla.

1. Merkitse $p = 2$.
2. Laske jakojäännös, kun n jaetaan p:llä. Merkitse jakojäännöstä r:llä.
3. Jos $r = 0$, palauta vastaus: n ei ole alkuluku. Jos $r \neq 0$ ja $p < n - 1$, kasvata p:n arvoa yhdellä ja palaa kohtaan 2. Jos $r \neq 0$ ja $p = n - 1$, palauta vastaus: n on alkuluku.

Merkitään jakojäännöksen palauttavaa funktiota j:llä, ts. $j(n, p)$ on jakojäännös, kun n jaetaan p:llä. Muotoillaan yo. algoritmi toistorakenteen muotoon. Silmukka katkeaa (toisto lakkaa), jos $j(n, p) = 0$, ts. n on jaollinen p:llä, tai viimeistään kun p on saavuttanut arvon $n - 1$. Jos silmukan jälkeen $j(n, p) \neq 0$, voidaan sanoa että n ei ole jaollinen millään itseään pienemmällä (ykköstä suuremmalla) luvulla, ts. n on alkuluku. Jos taas $j(n, p) = 0$, niin n on jaollinen p:llä (p:n sen hetkisellä arvolla), jolle $1 < p < n$, joten n ei ole alkuluku.

1. Merkitse $p = 2$ (oletetaan, että $n > 2$).
2. Laske $j(n, p)$.
3. Toista seuraavia käskyjä niin kauan kun $j(n, p) \neq 0$ ja $p < n - 1$:
    1. Merkitse $p = p + 1$.
    2. Laske $j(n, p)$.
4. Jos $j(n, p) = 0$, vastaa: n ei ole alkuluku. Jos $j(n, p) \neq 0$, vastaa: n on alkuluku.

Edelleen, yo. alkulukutestin voisi kirjoittaa omaksi funktiokseen, joka ottaa argumentikseen tutkittavan luvun n ja palauttaa esim. "on" tai "ei ole", tai 1 tai 0, riippuen siitä, onko n alkuluku vai ei.