LUTin C-ohjelmoinnin tyyliohje

1. Johdanto

Tämä on LUTin C-ohjelmointikurssien tyyliohje. Nämä ohjeet on tarkoitettu kurssin harjoitustyön valikkopohjaisen ohjelman tyyliohjeeksi auttamaan toimivan ja ymmärrettävän C-ohjelman tekemisessä. Tätä ohjetta voi soveltaa muissa vastaavissa projekteissa, mutta lähtökohtaisesti kaikilla ohjelmointiin vakavasti suhtautuvilla organisaatioilla ja projekteilla on omiin tarpeisiin sovitetut tyyliohjeet, joita tulee noudattaa ja soveltaa organisaation sisäisten ohjeiden mukaisesti.

1.1. Kurssin ohjelmien vaatimukset

LUTin C-ohjelmointikursseilla kaikkia ohjelmia koskee seuraavat vaatimukset:

1. Ohjelman on käännyttävä, mentävä automaattitarkastuksesta läpi ja toimittava tehtävänannon mukaisesti
2. Tiedostojen avaamisen ja muistin varaamisen yhteydessä on oltava virheenkäsittely
3. Kaikki ohjelman varaamat resurssit tulee vapauttaa ohjelman lopuksi kohdan 3.4 mukaisesti
4. Staattisesti varattavan taulukon koko on määriteltävä vakiolla käännösvaiheessa
5. Globaalit muuttujat ovat kiellettyjä
6. `goto`-lauseen käyttö on kielletty
7. Kaikissa palautettavissa tiedostoissa on oltava tämän ohjeen mukaiset alkukommentit
8. Kaikkien pää- ja aliohjelmien on päätyttävä `return`-käskyyn
9. Kurssilla käsittelemättömien kirjastojen käyttö on kielletty.

Nämä vaatimukset tulevat voimaan sen jälkeen, kun ne on käsitelty luennoilla ja niiden noudattamatta jättäminen johtaa työn korjaamiseen tai hylkäämiseen tilanteesta riippuen.

Kursseilla ei opeteta kaikkien C-kielen käskyjen käyttöä. Yllä olevassa listassa on nimetty kursseilla kielletyt ohjelmointirakenteet ja muita rakenteita/käskyjä saa käyttää, mutta vastuu niiden oikeasta käytöstä ja ohjelman selkeydestä on rakenteen käyttäjällä.

Muutoin tyyliohjeiden noudattaminen on kurssilla opetettavan ymmärrettävän ja ylläpidettävän ohjelmointityylin lähtökohta. Joissain tapauksissa näistä ohjeista tulee poiketa, jolloin asia mainitaan erikseen tehtävänannossa. Perustelut näille tyyliohjeille löytyy LUTin C-ohjelmointioppaasta ja tällä kurssilla hyvästä ohjelmointityylistä päättää kurssin vastuuopettaja.

2. Rakenne

2.1. Tiedostorakenne, yksi tiedosto

Tiedoston rakenne, kun ohjelma koostuu vain yhdestä tiedostosta:

1. Tiedoston alkukommentti
2. Otsikkotiedostojen sisällytys
3. Vakioiden määrittely
4. Uusien tietorakenteiden määrittely, erityisesti tietueet
5. Aliohjelmien esittelyt sisältäen muuttujien nimet
6. Pääohjelman koodi
7. Aliohjelmien koodi samassa järjestyksessä kuin esittelyssä

2.2. Tiedostorakenne, useista tiedostoista muodostuva ohjelma

Laajassa ohjelmassa on pääohjelma ja yksi tai useampi aliohjelma, esim. harjoitustyössä tyypillisesti vähintään neljä aliohjelmaa. Aliohjelmat tulee jakaa eri tiedostoihin alla olevan mukaisesti. Vakiot ja tietorakenteet määritellään ohjelmassa vain yhden kerran tyypillisesti omassa otsikkotiedostossa.

Pääohjelman sisältävät tiedoston rakenne:

1. Tiedoston alkukommentti
2. C-kielen otsikkotiedostojen sisällytys
3. Omien otsikkotiedostojen sisällytys
4. Pääohjelman koodi

Muiden C-tiedostojen rakenne:

1. Tiedoston alkukommentti
2. C-kielen otsikkotiedostojen sisällytys
3. Omien otsikkotiedostojen sisällytys
4. Aliohjelmien koodi samassa järjestyksessä kuin esittelyissä

Otsikkotiedostojen sisällytykset tehdään aina lähdekooditiedostoissa.

Jokaiseen C-tiedostoon tarvittaessa liittyvän otsikkotiedoston eli .h-tiedoston rakenne:

1. Tiedoston alkukommentti
2. Otsikkotiedoston ehdollinen sisällytys
3. Vakioiden määrittely
4. Tietorakenteiden määrittely
5. Aliohjelmien esittelyt sisältäen muuttujien nimet

2.3. Tiedoston alkukommentti

Kaikki kurssin ohjelmointitehtävät ovat henkilökohtaisia ja siksi palautettaviin ohjelmatiedostoihin tulee laittaa alkukommentti. Kurssin kaikkiin palautettaviin tiedostoihin tulee laittaa kommenttina seuraavat tiedot:

```
/* Päivämäärä, tekijä, tiedostonimi, tehtävä */
```

Kurssin harjoitustyössä tulee olla alla oleva yksityiskohtaisempi kommentti. Kommentissa mainitut koodiin vaikuttaneet lähteet ja henkilöt auttaa kurssihenkilöstöä ymmärtämään koodin taustat ja välttämään turhia vilppiepäilyjä.

```
/*****************************************************************************/
/* CT60A2500 C-ohjelmoinnin perusteet
 * Tekijä:
 * Opiskelijanumero:
 * Päivämäärä:
 * Kurssin oppimateriaalien lisäksi työhön ovat vaikuttaneet seuraavat
 * lähteet ja henkilöt, ja se näkyy tehtävässä seuraavalla tavalla:

 * Mahdollisen vilppiselvityksen varalta vakuutan, että olen tehnyt itse
 * tämän tehtävän ja vain yllä mainitut henkilöt sekä lähteet ovat
 * vaikuttaneet siihen yllä mainituilla tavoilla.
*/
/*****************************************************************************/
/* Tehtävä x, tiedoston nimi y */

/* eof */
```

2.4. Otsikkotiedoston ehdollinen sisällytys

Useista tiedostoista muodostuvan ohjelman otsikkotiedostoissa tulee olla ehdollinen sisällytys

1. Käytettävän vakion nimi on sama kuin tiedostonimi suuraakkosilla, esim. `tiedosto.h` -tiedoston vakion tulee olla `TIEDOSTO_H`
2. Ehdollinen sisällytys muodostuu seuraavista käskyistä: `#ifndef - #define - #endif`

2.5. Ohjelman rakenne

Pää- ja aliohjelmat muodostuvat lähtökohtaisesti seuraavista osista tilanteen mukaan:

1. Muuttujien määrittelyt ja alustukset
2. Tietojen kysyminen käyttäjältä
3. Toiminnallinen osuus, laskenta
4. Tulosten tulostaminen käyttäjälle
5. Lopetusrutiinit, muistin vapautus ja käyttäjän informointi ohjelman loppumisesta

2.6. Pääohjelman rakenne

Valikkopohjainen ohjelma perustuu toistorakenteen sisällä olevaan valintarakenteeseen

1. Toistorakenteena käytetään `do-while` -rakennetta
2. Ohjelman normaali lopetus päättyy `while`-ehdon kautta.
3. Ohjelma voi päättyä aiemmin normaalin virheenkäsittelyn seurauksena `exit`-käskyllä
4. Ohjelman valikon käsittely tapahtuu omassa valikko-aliohjelmassa
5. Valintarakenteena käytetään monihaaraista `if`-rakennetta
6. Valintarakenteen valinnat käydään järjestyksessä alkaen valinnasta 1 valintaan N, jonka jälkeen on valinta 0 eli ohjelman lopetus
7. Jokaisen valinnan kohdalla tavoite on tehdä kaikki valintaan liittyvät toimenpiteet alusta loppuun asti, esim. tiedoston lukemisen yhteydessä kysytään luettavan tiedoston nimi, luetaan tiedosto ja kerrotaan käyttäjälle, että lukeminen onnistui
8. Mikäli valinnan suorittaminen edellyttää tiettyjä tietoja, esim. linkitettyä listaa, tarkistetaan tämän tiedon olemassaolo ennen siihen liittyvien operaatioiden suorittamista ja tarvittaessa kerrotaan käyttäjälle, ettei operaatioita voida suorittaa tiedon puuttumisen vuoksi. Käyttäjätiedotteissa on pyrittävä kertomaan selkeästi ongelma ja miten se voidaan ratkaista
9. Valintarakenteen viimeinen haara on `else`-haara, jossa käsitellään tuntemattomat valinnat
10. Valintarakenteen jälkeen viimeinen käsky ennen toistorakenteen lopetusehtoa on tyhjän rivin tulostus
11. Ohjelman kaikki lopetusrutiinit ovat toistorakenteen jälkeen ennen pääohjelman loppua

Pääohjelman ja valintarakenteen standardifraasit ovat seuraavat:

1. `"Tuntematon valinta, yritä uudestaan.\n"`
2. `"Lopetetaan.\n"`
3. `"Kiitos ohjelman käytöstä.\n"`

Valinta-rakenteessa tyypillisiä ohjeita ja tiedotteita käyttäjälle ovat mm. seuraavat:

1. `"Anna luettavan tiedoston nimi: "`
2. `"Anna kirjoitettavan tiedoston nimi: "`
3. `"Ei analysoitavaa, lue tiedosto ennen analyysiä.\n"`
4. `"Ei kirjoitettavia tietoja, analysoi tiedot ennen tallennusta.\n"`
5. `"Tiedosto '%s' luettu.\n"`
6. `"Tiedosto '%s' kirjoitettu.\n"`
7. `"Tiedosto '%s' luettu ja tulostettu."`
8. `"Tiedot luettu linkitettyyn listaan."`
9. `"Muisti vapautettu."`

2.7. valikko-aliohjelma

Valikko-pohjaisen ohjelman valikko-aliohjelma tulee toteuttaa seuraavalla tavalla:

1. Aliohjelma ei saa parametrejä ja se palauttaa käyttäjän valinnan kokonaislukuna
2. Valikon jokainen rivi tulostetaan omalla `printf`-funktiolla, joka päättyy rivinvaihtoon
3. Käyttäjän valinta luetaan `scanf`-funktiolla ja sen jälkeen puskurista poistetaan rivinvaihtomerkki
4. Käyttäjän valinnat numeroidaan alkaen luvusta 1 ja viimeisenä on valinta Lopeta luvulla 0, numeron jälkeen on)-merkki ja välilyönti

valikko-aliohjelman standardifraasit ovat seuraavat:

1. `"Valitse haluamasi toiminto:\n"`
2. `"Anna valintasi: "`

2.8. Aliohjelmat

Aliohjelmat jakavat ohjelman pienempiin loogisiin kokonaisuuksiin, mahdollistavat uudelleenkäytön ja lisäävät ymmärrettävyyttä. Aliohjelmiin liittyy seuraavat yleisohjeet:

1. Uusi aliohjelma tulee tehdä, kun sama/vastaava toiminta tehdään ohjelmassa monta kertaa tai ohjelmaan lisätään uusi erillinen toiminnallisuus kuten tiedoston luku tai kirjoitus
2. Ohjelman toiminnallisuuden lisäys voi kasvattaa ohjelmaa ja tehdä siitä hankalasti ymmärrettävän, jolloin loogisia kokonaisuuksia tulee siirtää omiin aliohjelmiin
3. Uutta aliohjelmaa ei tule tehdä, jos se ei tuo lisäarvoa.

Aliohjelmien tiedonvälitys

1. Tiedot aliohjelmiin välitetään parametrien avulla
2. Pääohjelma ja kaikki aliohjelmat päättyvät `return`-käskyyn
3. Lähtökohtaisesti aliohjelmista tulee palauttaa hyödyllistä tietoa
    a. Linkitetyn listan tyhjennys -aliohjelmasta kannattaa palauttaa `NULL`
    b. Tiedoston luku -aliohjelmasta voi palauttaa luettujen rivien määrän
    c. Aliohjelmasta voi palauttaa statuskoodin, ts. ok tai virhekoodi
4. Globaalit muuttujat ovat kiellettyjä
5. Globaalien vakioiden käyttö on suositeltavaa.

2.9. Nimeäminen

Ohjelmoinnissa tiedostot ja tunnukset tulee nimetä selkeästi ja johdonmukaisesti.

Yleisesti

1. Skandinaavisia merkkejä (å, ä, ö, Å, Ä, Ö) ei tule käyttää tiedostojen ja tunnusten nimissä. Tyypillisesti nämä korvataan a tai o -kirjaimella
2. Tunnusten tulee olla kuvaavia ja yksikäsitteisiä
3. Samassa nimiavaruudessa olevat tunnukset tulee nimetä toisistaan poikkeavilla nimillä, esim. kiintoarvoilla, muuttujilla ja aliohjelmilla on oltava eri nimet

Tiedostot

1. Pääohjelman lähdekooditiedoston ja suoritettavan tiedoston nimien tulee olla samat
2. Viikkotehtävät tulee nimetä luennon ja tehtävän perusteella, esim. `L1T1.c`, `L7T2.c`
3. Kirjastotiedostojen nimistä tulee käydä ilmi, mihin tehtävään ne liittyvät, esim. `L5T1Kirjasto.c`

4. Aliohjelmia sisältävien tiedostojen otsikko- ja ohjelmatiedostojen tulee olla saman nimisiä siten, että vain tiedostojen tiedostopäätteet eroavat, esim. `tiedosto.c` ja `tiedosto.h`
5. Harjoitustyötiedostot tulee nimetä tehtävänannon mukaisesti
6. Tentissä tiedostot tulee nimetä tentin ohjeiden mukaisesti

Vakiot

1. Esikääntäjällä (`#define`) ja `enum`:lla määritellyt vakiot kirjoitetaan kaikki kirjaimet suurakkosilla, esim. `#define LKM 10`
2. Vakioita tulee käyttää mm. staattisesti varattujen taulukoiden kokojen määrittelyyn

Muuttujat

1. Muuttujien nimet kannattaa määritellä käyttäen unkarilaista notaatiota, jolloin muuttujan tietotyyppi näkyy sen nimestä. Tällä kurssilla keskitytään yleisimpiin perustietotyyppeihin eikä harvinaisempia tietotyyppejä huomioida
2. Käytettävät tietotyypit ovat seuraavat
    a. i – int, kokonaisluku
    b. l – long, pitkä kokonaisluku
    c. f – float, liukuluku
    d. d – double, kaksoistarkkuuden liukuluku
    e. c – char, merkki
    f. a – array, taulukko ottamatta kantaa taulukon tietoalkioiden tyyppiin, joka voi näkyä muuttujan nimessä, esim. `aNimi`, `aNumerot`
    g. p – pointer, osoitin ottamatta kantaa osoitettavaan tietotyyppiin, esim. `pAlku`. Yleinen liukuri-osoitin on tyypillisesti nimeltään `ptr`
3. Muuttujien nimet kannattaa muodostaa systemaattisesti niiden kuvaaman tiedon perusteella, esim. `iLukumaara`, `aNimiEtu`, `aNimiSuku`, `dPaino`, `dPainoNetto`, `dPainoMax`

Tietueet

1. Tietue tulee nimetä pienaakkosilla, esim. `struct henkilo`
2. Tietueesta tulee määritellä uusi tietotyyppi, joka nimi on sama kuin tietueella, mutta kirjoitettuna suurakkosilla, esim. `typedef struct henkilo HENKILO;`

Aliohjelmat

1. Aliohjelman nimen tulee kertoa, mitä aliohjelmassa tapahtuu ja tarvittaessa käytetään useita sanoja, esim. `analysoiTiedot`, `kirjoitaTiedosto`, `tulostaTiedot`
2. Yhdestä sanasta muodostuvat aliohjelmanimet tulee kirjoittaa kaikki kirjaimet pienellä ja jos sanoja on useita, seuraavien sanojen ensimmäiset kirjaimet kirjoitetaan suuraakkosilla, esim. `valikko`, `kysyNimi`, `lueTiedosto`

Yksikirjaimiset ja mitäänsanomattomat tunnukset/nimet ovat kiellettyjä

1. Kiellettyjä tunnuksia ovat mitään sanomattomat tunnukset, esim. `a`, `b`, `c` ja `a1`, `a2`, `a3`
2. C-ohjelmien vakiintuneet muuttujat esim. `i`, `j` ja `k` ovat hyväksyttäviä
3. Epävarmoissa tapauksissa kannattaa käyttää kuvaavia nimiä yllä olevan mukaisesti

2.10. Näkyvyys

Tunnusten näkyvyyteen liittyen tulee muistaa seuraavat asiat:

1. Globaalit muuttujat ovat kiellettyjä
2. Muuttujat määritellään lokaaleina ohjelmissa/koodilohkoissa
3. Vakiot, tietueet ja aliohjelmat määritellään globaaleina

3. Perusoperaatiot

3.1. Tietojen kysyminen käyttäjältä

Tiedot tulee kysyä käyttäjältä seuraavilla tavoilla:

1. Numerot luetaan `scanf`:lla
2. Merkit luetaan `scanf`:lla, puskuriin jäävä rivinvaihtomerkki tulee poistaa
3. Sanat luetaan `scanf`:lla
4. Lauseet luetaan `fgets`:llä, merkkijonoon sisältyvä rivinvaihtomerkki tulee poistaa

Tietoja kysyttäessä niille pitää olla varattuna muuttujat, joiden tulee olla riittävän suuria luettavalle tiedolle ja muistialue pitää olla käytettävissä niin pitkään kun muuttujia käytetään. Tämä tulee huomioida erityisesti luettaessa tietoja aliohjelmissa eli muisti pitää tällöin varata joko dynaamisesti tai kutsuvan ohjelman puolella.

3.2. Tiedostonkäsittely

Tekstitiedostonkäsittely tehdään omassa aliohjelmassa

1. Aliohjelma saa parametrinä osoittimen luettavan tiedoston nimeen
2. Avaa tiedosto ja tarkista sen onnistuminen normaalilla virheenkäsittelyllä
3. Tiedoston lukeminen
    a. Tiedoston lukemiseen käytetään tyypillisesti `fgets`-funktioita. Merkkijonojen pilkkominen alkioihin voidaan tehdä esim. `strtok`-funktiolla ja merkkijonoja voi muuttaa numeroiksi esim. `atoi` ja `atof`-funktioilla
4. Tiedoston kirjoittaminen
    a. Kirjoitettava tiedosto avataan tyypillisesti kirjoitustilassa, joka hävittää aiemman saman nimisen tiedoston, jos sellainen on
    b. Tiedoston kirjoittamiseen käytetään tyypillisesti `fprintf`-funktiota
5. Käytön jälkeen tiedosto on suljettava samassa aliohjelmassa

Mikäli samat tiedot voidaan kirjoittaa tiedostoon ja tulostaa näytölle, tulee varsinainen tulostaminen tehdä yhdessä aliohjelmassa. Tällöin tulostuksen tekevä aliohjelma saa parametrina tavoitteesta riippuen joko osoittimen tulostettavaan tiedostoon sen avaamisen jälkeen, tai osoittimen `stdout`-standarditulostevirtaan.

Tiedostonkäsittelyn tyypillisiä ohjeita ja tiedotteita käyttäjälle ovat mm. seuraavat:

1. `"Tiedosto 'tiedoston_nimi' luettu."`
2. `"Tiedosto 'tiedoston_nimi' luettu ja tulostettu."`
3. `"Tiedosto 'tiedoston_nimi' kirjoitettu."`

3.3. Linkitetyn listan luominen

Luotaessa linkitetty lista tiedoston riveillä olevasta datasta

1. Tiedosto luetaan omassa aliohjelmassa ja sen tiedoista muodostetaan lukemisen yhteydessä linkitetty lista. Aliohjelma saa parametreina luettavan tiedoston nimen merkkijonona ja osoittimen listan alkuun
2. Mikäli lista on jo olemassa, vapauta varattu muisti tyhjentämällä lista
3. Lue tiedostosta yksi rivi tietoa kerrallaan ja lisää tiedot listaan omassa aliohjelmassaan. Aliohjelma saa parametreina osoittimen listan ensimmäiseen alkioon ja lisättävät tiedot sisältävän merkkijonon osoitteen
4. Yhdellä rivillä olevat tiedot laitetaan aina yhden tietueen tiedoksi siten, että rivin jokaisesta tietoalkiosta tulee oma jäsenmuuttuja

5. Linkitetyssä listassa jokainen tietue sisältää tieto-jäsenmuuttujien lisäksi seuraavaan listan alkioon osoittava osoittimen, jonka nimi on `pSeuraava`
6. Varaa tietueen tarvitsema muisti `malloc`-funktiolla ja normaalilla virheenkäsittelyllä
7. Kaikki riveillä olevat tietoalkiot muutetaan niiden luonnollisiksi tietotyypeiksi. Näin ollen kokonaisluvut muutetaan kokonaislukutyypeiksi (`int`, `long`, …), desimaaliluvut liukuluvuiksi (`float`, `double`, …), nimet merkkijonoiksi, päivämäärät aika-tiedoksi (`struct tm`, `time_t`, …) jne.
8. Listaan lisäys -aliohjelmasta palautetaan osoitin listan ensimmäiseen alkioon tai `NULL`-osoitin virheen merkkinä
9. Kun kaikki tiedoston rivit on lisätty linkitettyyn listaan, palataan tiedoston luku -aliohjelmasta ja palautetaan listan ensimmäisen alkion osoite

Mikäli linkitetty lista luodaan käyttäjän antamista tiedoista, toimitaan edellä olevan ohjeen mukaisesti sillä erolla, että tietojen tiedostosta lukemisen sijaan ne kysytään käyttäjältä.

3.4. Varattujen resurssien vapauttaminen

C-ohjelmissa varattavat resurssit kuten muisti ja tiedostokahvat on vapautettava ennen ohjelman päättymistä

1. Dynaamisesti varattu muisti pitää vapauttaa `free`-funktiolla ja asettaa osoitin `NULL`:ksi
2. Tiedostokahvat tulee vapauttaa sulkemalla avatut tiedostot samassa aliohjelmassa missä ne avattiin

3.5. Tietojen analysointi

Tietojen analyysi tehdään tyypillisesti linkitetyssä listassa oleville tiedoille

1. Analyysi-aliohjelma saa parametrina osoitteen listan ensimmäiseen alkioon sekä osoitteen tulostietorakenteeseen. Tulostietorakenne vaihtelee tilanteen mukaan ollen tyypillisesti yksi tietue tai linkitetty lista
2. Mikäli tulostietorakenne on dynaaminen ja jo olemassa, vapauta se
3. Tee analyysi ja sijoita tulokset tulostietorakenteeseen sekä varaa tarvittaessa muistia normaalilla virheenkäsittelyllä
4. Etsittäessä datasta tiettyjä arvoja, esim. minimi tai maksimi, tulee etsintä aloittaa datasetin ensimmäisen alkion arvoilla
5. Mikäli datassa on useita ehdon täyttäviä arvoja, esim. useita yhtä suuria minimi-/maksimiarvoja, valitaan näistä
    1. erikseen mainitun ehdon täyttävä arvo, jos tällainen ehto on olemassa
    2. vanhin, jos datassa on aikaleima
    3. alkuperäisen datan ensimmäinen arvo (rivinumeron mukaan)
6. Mikäli tehtävässä käytetään datasetin ominaisuuksia, tulee ne selvittää datasta, esim. alkioiden lukumäärä tai ensimmäisen/viimeisen alkion aikaleima
7. Kaikki analyysit tulee suorittaa alkuperäisissä yksiköissä ja mahdollinen tulosten pyöristys tehdään vasta muotoiltaessa lopullisia tulosteita
8. Palauta tulostietorakenteen osoite kutsuvaan ohjelmaan

3.6. Virheenkäsittely

Virheenkäsittely pitää toteuttaa aina tiedostonkäsittelyn yhteydessä tiedostoa avattaessa ja muistia varattaessa

1. Mikäli operaatio ei onnistu, kerrotaan käyttäjälle ongelmasta `perror`-funktiolla ja virheilmoituksella
2. Ohjelman suoritus lopetetaan `exit(0)`-käskyllä

Tyypillisimmät virheilmoitukset ovat seuraavat:

1. `"Muistinvaraus epäonnistui, lopetetaan"`
2. `"Tiedoston avaaminen epäonnistui, lopetetaan"`
3. `"Merkkijonon pilkkominen epäonnistui, lopetetaan"`

`perror`-funktiota käytettäessä virheilmoitukseen ei laiteta rivinvaihtoa, sillä sen perään tulee `perror`:n standardi virheilmoitus.

3.7. Ohjelman kääntäminen ja Makefile

Useiden tiedostojen kääntämiseen tulee käyttää make:a ja Makefile:ä.

1. Jokainen lähdekooditiedosto on käännettävä objektitiedostoksi omalla käskyllä. Objektitiedostojen nimien tulee olla samat kuin lähdekooditiedostojen
2. Suoritettava koodi tehdään omalla käskyllä käännetyistä objektitiedostoista
3. Käännettäessä on käytettävä seuraavia optiota: `-Wall`, `-pedantic`, `-std=c99`
4. `Makefile`:ssä on näyttävä tiedostojen riippuvuudet
5. `Makefile`:ssä tulee olla alkukommentti sisältäen vähintään tekijän ja päivämäärän
6. Harjoitustyön suoritettavan tiedoston tulee olla nimeltään HT

3.8. Ylimääräiset koodirivit

Ohjelmassa ei tule olla ylimääräisiä koodirivejä, jotka eivät tee mitään, kumoavat toisensa tai joita ei voi koskaan saavuttaa. Tällaisia ovat mm. seuraavat:

1. Muuttujat, vakiot, aliohjelmat ja tietorakenteet, jotka määritellään, mutta joita ei käytetä sen jälkeen
2. Käskyjen `break`, `continue`, `return` ja `exit` jälkeen samassa koodilohkossa olevat käskyt

3.9. Yksittäisiä huomioita

1. Ohjelmarivit on sisennettävä loogisesti samalla tyylillä koko ohjelmassa. Suositus on sisentää koodia aina neljää (4) välilyöntiä, joka onnistuu useimmissa koodieditoreissa sarkainnäppäimellä
2. Tulosteissa rivinvaihtomerkkien tulee lähtökohtaisesti olla merkkijonon lopussa tai omina erillisinä käskyinä
3. Lähtökohtaisesti toistorakenteen tulee päättyä normaalisti, jotta sen jälkeen olevat lopetusrutiinit tulevat suoritettua. Normaali lopetus tarkoittaa, että kaikki läpikäytävät arvot on käsitelty tai ohjelman suoritus päättyy poikkeustenkäsittelijään
4. Askeltavaa toistoa (`for`) käytetään, kun kierrosmäärä on tiedossa etukäteen. `while`-rakennetta käytetään, kun läpikäytävien tietojen lukumäärä ei ole tiedossa etukäteen tai lopetusehtoja on useita
5. `goto`-käskyn käyttö on kielletty
6. Rekursiota ei tule käyttää, ellei tehtävänannossa ole niin erikseen kehotettu
7. Mikäli ohjelman suorituksessa tulee ongelmia, tulee ohjelman kertoa käyttäjälle selkeästi, mikä ongelma on ja miten sen voi ratkaista. Tämä koskee erityisesti virheiden ennaltaehkäisyä ja virheenkäsittelyä, ks. Kohdat 2.6 ja 3.6.