Lukijalle

Tähän monisteeseen on kerätty Ohjelmointi-98 kurssin harjoitustyöpäiväkirja sekä harjoitustyön teko-ohjeet .

Tämä ei ole mikään luentomoniste, vaan kokoelma papereita, joita tarvitaan Ohjelmointi-kurssilla. Paperit on kasattu yhdeksi monisteeksi, jotta ne olisivat helpompi pitää aina mukana. Lisäksi koko on saatu näin miellyttävämmäksi.

Tästä monisteesta lukijan edellytetään erityisesti lukevan **harjoitustyön teko-ohjeet**. Lisäksi tämän monisteen pääteohjaus-luvuissa käsitellään Turbo-C:tä (tai Borland C:tä) kun varsinaiset monisteet keskittyvät pelkästään standardi ANSI-C:hen.

Varsinainen luentomoniste on omana monisteenaan: *Ohjelmointi++*. Lisäksi tarvitaan vielä luentomonisteen esimerkki –moniste: *Listaukset*. Edelleen muistutetaan jonkin C- ja C++ –kieliä perusteellisesti käsittelevän kirjan hyödyllisyydestä.

Huomattakoon tästä monisteesta (niinkuin allekirjoittaneen monisteista yleensäkin), että sisällysluetteloa kannattaa käyttää ahkeraan muistilistana ja koko monisteen “pikaoppaana”.

Palokassa 6.1.1998

Vesa Lappalainen

1. Harjoitustyön ohjauspäiväkirja

Tämä päiväkirja pidetään aina mukana tultaessa ohjaustilaisuuteen.

Nimi: ________________________________ Pääaine: ________________________

Sotu: _______________________

Aikaisemmat ATK-kurssit: ____

Harjoitustyön aihe (saa keksiä myös itse): ___________________________________

____

____

____

____

1. Esisuunnitelma

Valmis viimeistään __.__.1998

Pvm Muutosvaatimukset ohjaajan kuitaus

__.__ ____

__.__ ____

__.__ ____

__.__ ____

__.__ ____

__.__ ____

__.__ ____

__.__ ____

Harjoitustyön vaihe hyväksytty: ___.___.1998

Ohjaajan kuittaus ____

2. Käyttöohje tai suunnitelma

Valmis viimeistään __.__.1998

Pvm	Muutosvaatimukset	ohjaajan kuittaus
__.__		
__.__		
__.__		
__.__		
__.__		
__.__		
__.__		
__.__		
__.__		
__.__		
__.__		
__.__		
__.__		
__.__		
__.__		
__.__		
__.__		
__.__		
__.__		
__.__		
__.__		
__.__		
__.__		

Harjoitustyön vaihe hyväksytty: ___.___.1998

Ohjaajan kuittaus ____

3. Ohjelman valikkorakenne (menut)

Valmis viimeistään __.__.1998

Pvm Muutosvaatimukset ohjaajan kuittaus

__.__ ____________________
__.__ ____________________
__.__ ____________________
__.__ ____________________
__.__ ____________________
__.__ ____________________
__.__ ____________________
__.__ ____________________
__.__ ____________________
__.__ ____________________
__.__ ____________________
__.__ ____________________
__.__ ____________________
__.__ ____________________
__.__ ____________________
__.__ ____________________
__.__ ____________________
__.__ ____________________
__.__ ____________________
__.__ ____________________
__.__ ____________________
__.__ ____________________
__.__ ____________________

Harjoitustyön vaihe hyväksytty: ___.___.1998

Ohjaajan kuittaus ____________________

4. Tietorakenteet ja oliosuunnittelu

Valmis viimeistään __.__.1998

Pvm Muutosvaatimukset ohjaajan kuittaus

__.__ ____________________

__.__ ____________________

__.__ ____________________

__.__ ____________________

__.__ ____________________

__.__ ____________________

__.__ ____________________

__.__ ____________________

__.__ ____________________

__.__ ____________________

__.__ ____________________

__.__ ____________________

__.__ ____________________

__.__ ____________________

__.__ ____________________

__.__ ____________________

__.__ ____________________

__.__ ____________________

__.__ ____________________

__.__ ____________________

__.__ ____________________

__.__ ____________________

__.__ ____________________

Harjoitustyön vaihe hyväksytty: ___.___.1998

Ohjaajan kuittaus ____________________

5. Ohjelman runko

Valmis viimeistään __.__.1998

Pvm Muutosvaatimukset ohjaajan kuittaus

__.__ ____________________

__.__ ____________________

__.__ ____________________

__.__ ____________________

__.__ ____________________

__.__ ____________________

__.__ ____________________

__.__ ____________________

__.__ ____________________

__.__ ____________________

__.__ ____________________

__.__ ____________________

__.__ ____________________

__.__ ____________________

__.__ ____________________

__.__ ____________________

__.__ ____________________

__.__ ____________________

__.__ ____________________

__.__ ____________________

__.__ ____________________

__.__ ____________________

__.__ ____________________

Harjoitustyön vaihe hyväksytty: ___.___.1998

Ohjaajan kuittaus ____________________

6. Tiedoston käsittely ja moduleihin jako

Valmis viimeistään __.__.1998

Pvm Muutosvaatimukset ohjaajan kuittaus

__.__ ______________________________

__.__ ______________________________

__.__ ______________________________

__.__ ______________________________

__.__ ______________________________

__.__ ______________________________

__.__ ______________________________

__.__ ______________________________

__.__ ______________________________

__.__ ______________________________

__.__ ______________________________

__.__ ______________________________

__.__ ______________________________

__.__ ______________________________

__.__ ______________________________

__.__ ______________________________

__.__ ______________________________

__.__ ______________________________

__.__ ______________________________

__.__ ______________________________

__.__ ______________________________

__.__ ______________________________

__.__ ______________________________

Harjoitustyön vaihe hyväksytty: ___.___.1998

Ohjaajan kuittaus ______________________________

7. Toimiva ohjelma

Valmis viimeistään __.__.1998

Pvm Muutosvaatimukset ohjaajan kuittaus

__.__ ____

__.__ ____

__.__ ____

__.__ ____

__.__ ____

__.__ ____

__.__ ____

__.__ ____

__.__ ____

__.__ ____

__.__ ____

__.__ ____

__.__ ____

__.__ ____

__.__ ____

__.__ ____

__.__ ____

__.__ ____

__.__ ____

__.__ ____

__.__ ____

__.__ ____

__.__ ____

Harjoitustyön vaihe hyväksytty: ___.___.1998

Ohjaajan kuittaus ____

2. Harjoitustyön teko-ohje

2.1 Harjoitustyökansio

Parasta olisi heti alkuun tehdä itselleen harjoitustyökansio, jossa on tämä työkirja ja selvät välilehdet kutakin harjoitustyön vaihetta varten. Lisäksi kansioon voi tehdä taskun harjoitustyön levykkeille. Levykkeitä kannattaa olla vähintään 1 varsinainen ja 1 varakopio.

2.2 Ohjaukset

Harjoitustyön ohjausta annetaan mielihyvin kaikille niille, jotka tulevat apua ja/tai ohjausta pyytämään.

Koska ohjattavia on paljon, pitää ohjaukseen varata aika merkitsemällä oma nimensä oven vieressä olevaan lukujärjestykseen. VL:n luokse voi tulla myös muulloin, mikäli hän sattuu olemaan vapaana.

Ohjaukseen tultaessa pitää olla varustautunut seuraavasti:

- valmiilla kysymyksillä joihin vastausta haetaan
- mielellään ohjelmalistauksen kanssa (harjoitustyökansio)
- aina alkuperäisen suunnitelman ja tietorakenteen kuvan kanssa
- mikäli ohjelmaa täytyy katsoa levykkeeltä, niin sitten täytyy olla vastaava tiedostolista (`OHJ.SIS`) kuin myöhemmin olevassa mallissa
- harjoitustyöpäiväkirjaan on merkitty viimeiset palautuspäivät
- harjoitustyöpäiväkirjan kanssa, johon on valmiiksi merkitty päivämäärä
- kynä esillä
- kaikki em. kaivetaan esiin jo ennen ohjaajan huoneeseen astumista; ei ole ystävällistä kaivaa kassiaan 5 minuuttia ohjausajasta ja näin viedä muiden ohjattavien aikaa
- ryhmätyössä kaikki ryhmän jäsenet mukana
- huoneeseen tullaan sisälle koputuksen jälkeen vasta kun siihen on annettu lupa

Ohjauksessa:

1. ei ole mitään pelättävää
2. esitettävä kritiikki ja korjausehdotukset on tarkoitettu positiivikseksi opetukseksi
3. kirjoitetaan päiväkirjaan oma-aloitteisesti ohjaajan esittämät muutokset
4. ohjaaja tarkistaa muutosehdotukset samoiksi joita hän tarkoitti ja kuittaa ne omikseen

5. kun muutokset on tehty, kuittaa ohjaaja myös tämän

2.3 Harjoitustyö ryhmätyönä

Kurssilla on mahdollisuus tehdä harjoitustyö myös ryhmätyönä. Ryhmäläisillä voi olla kaikilla joko sama aihe jota he tekevät yhdessä tai sitten kullakin oma aihe, jota yhdessä voidaan työstää. Esimerkiksi yhden tekemät aliohjelmat kelpaavat muillekin, yksi ymmärtää ja selostaa muille. Kaikki voivat testata toistensa suunnitelmia ristiin.

Erillisissä töissä on se etu, että jonkin ryhmäläisen lopettaessa voidaan työ saattaa yksin loppuun.

2.4 Mikä on harjoitustyö

Harjoitustyö `EI` ole **työnäyte**!

2.4.1 Oppimistilaisuus

Harjoitustyö on **oppimistilaisuus**, jossa opiskelija yrittää omaamansa tiedon pohjalta tehdä jotakin. Tätä yritystä näytetään työn ohjaajalle, joka sitten tutkii mikä työssä on oikein ja mikä väärin, huonosti tai kömpelösti toteutettu. Näin jo opiskelun alkuvaiheessa päästään korjaamaan vääriä tottumuksia, joita jälkeenpäin piintyneinä tapoina voi olla mahdoton muuttaa.

Tämän takia harjoitustyötä **PITÄÄ** käydä esittämässä ohjaajille riittävän usein. Punakynämerkintä EI OLE HÄPEÄ, vaan osoitus siitä, mitä pitäisi pohtia uudelleen (ja osoitus siitä, että ohjaaja on kiinnostunut opiskelijan työstä!).

2.4.2 Kirjoita heti oikealla tyylillä

Kun kirjoitat harjoitustyötä, käytä heti oikeaoppista sisennystä, muuttujien nimien valintaa yms. On liian aikaa vievää ensin kirjoittaa jonkinlainen sotku ja työn loppuvaiheessa käyttää 10 tuntia tyylin muuttamiseen.

Kun oikean tyylin oppii, on sillä aivan yhtä nopea kirjoittaa kuin suttutyylilläkin!

2.5 Mitä pitää pohtia

2.5.1 Ohjelman helppokäyttöisyys ja loogisuus

Ohjelman tulisi olla sellainen, että sitä osaa käyttää kuka tahansa. Ammattimaisempaan käyttöön tehty ohjelma voi sisältää vähemmän ohjeita kuin yleiseen käyttöön tehty ohjelma.

2.5.2 Kuka käyttää ohjelmaa

Harjoitustyön tehtävän määritys on varsin epämääräinen. Jotta aiheesta saa järkevän, kannattaa ennen muuta jäsentämistä pohtia sitä, missä vastaavalla ohjelmalla olisi käyttöä.

Esimerkiksi olkoon tehtävänä tehdä ohjelma, jolla voidaan tutkia elokuvateatterin kävijöiden ikä ja sukupuolijakaumaa.

Missä tällaisella ohjelmalla olisi käyttöä? Tietysti elokuvateatterin johto voisi tämän perusteella valita mitä teatterissa kannattaa esittää ja milloin näytöksiä pidetään.

Mistä tiedot saadaan? Tiedot voitaisiin saada suoraan ovelta kävijöiden mennessä teatteriin. Mikro olisi ovella ja jokainen kävijä näppäilisi kysytyt tiedot koneeseen. Tällöin tämän osan ohjelmasta tulisi olla todella idioottivarman. Kun kaikki ovat menneet sisälle, kävisi teatterin johto jollakin salasanalla lopettamassa syötön ja ottaisi tarvitsemansa analyysin sekä tallettaisi tulokset tiedostoon myöhempää käyttöä varten!

2.5.3 Yleisyys ja muutettavuus

Esimerkiksi tehtävänä naisten telinevoimistelun pistelasku. Telineet, niiden nimet ja niiden määrä täytyy olla jälkeenpäin helposti muutettavissa, koska tuskinpa muuten ohjelmassa on isoa ero siinä lasketaanko miesten vai naisten kilpailua.

2.6 Oma aihe

Niille, joilla ei ole esittää omaa ehdotusta harjoitustyön aiheeksi, jaetaan äärimmäisessä hätätilassa valmiita aiheita.

2.6.1 Aihe keksitään itse

Opiskelijan kannalta on tietysti motivoivampaa valita itse aiheensa, josta haluaa tehdä ohjelman. Aihe voi olla esimerkiksi opettajilla jokin opetusohjelma, urheilijoilla jokin pistelasku tai päiväkirja, koti-isillä reseptivarasto jne.

Ehdotettua aihetta voidaan sitten ohjaajien kanssa muokata vastaamaan tasoltaan kurssin vaatimuksia.

Tavoitteena on ensin saada itseä miellyttävä aihe, jonka käyttötarpeen, toiminnan ja hyödyllisyyden ymmärtää. Tämän jälkeen aihe kiinnitetään ja ruvetaan harjoittelemaan ohjelman suunnittelua ja ohjelmointia tähän aiheeseen.

2.6.2 Vaatimukset

Harjoitustyössä pitäisi saada harjoiteltua seuraavat asiat:

- ohjelman suunnittelu
- tietorakenteiden ja olioluokkien suunnittelu
- ohjelman jako osiin (aliohjelmat, ohjelmamodulit)
- tiedostojen käyttö

– dynaamiset tietorakenteet (osoittimet)
– tietorakenteen läpikäyminen (haku, etsiminen)

Siis aihe tulisi valita siten, että siinä käsiteltävä aineisto pitäisi ohjelman käyttökertojen välillä tallettaa tiedostoon. Aineiston tarpeen pitäisi olla niin suuri, ettei sille välttämättä osata sanoa ennakolta tarkkaa ylärajaa. Tyypillisiä tällaisia ohjelmia ovat erilaiset rekisteriohjelmat, esim. mallina oleva kerhon jäsenrekisteri.

2.7 Harjoitustyön vaiheet

Huomattakoon, ettei harjoitustyön vaiheilla ja malliharjoitustyön numeroilla tai Listaus-monisteen luvun numeroilla ole yhteyttä keskenään (paitsi isompi numero on myöhempi vaihe). Seuraavassa on harjoitustyön vaiheiden, monisteen ja malliohjelman versioiden välinen yhteys:

1. Esisuunnitelma (Ohjelmointi, luku 2)

Harjoitustyön 1. vaihe on esisuunnitelma, eli tutkia tehtävän määritys tarkemmin ja yrittää määritellä siitä minkälainen ohjelman tulisi olla. Tässä tehdään monisteen Kerho-ohjelman mallin mukainen (tai parempi) suunnitelma siitä mitä ohjelma tekee.

Erityisesti kirjoitetaan kaikki ne kohdat, joita **näyttöön tulostuu**, mitä **käyttäjä vastaa** ja miten ohjelman tulisi eri vastauksiin reagoida! Siis

```
Anna hetu >111375-111A[RET][RET]
Henkilötunnuksessa päivämäärä väärin!
Anna hetu >
```

ei missään tapauksessa pelkkiä koneen tekemisiä:

```
Anna hetu >                                  VÄÄRIN TEHTY!
Henkilötunnuksessa päivämäärä väärin!
Anna hetu >
```

Jo tämä vaihe kannattaa kirjoittaa esimerkiksi Borland-C editorilla, jolloin sitä on helppo korjailla ohjaajien toivomusten mukaisesti. Myös se mitä tiedostoon talletetaan ja minkä näköisenä, tulisi esittää!

Koska Kerho-ohjelma jaetaan kurssilla mallina, saattaa ohjaaja tehdä suunnitelmaan muutoksia tai lisäyksiä, jottei työ synny pelkästään vaihtamalla malliohjelmassa sanoja Kerho sanoiksi Kauppa ja sanoja Jäsen sanoiksi Tuote.

Vaikka hieno näytönkäsittely yms. kuuluvatkin oikeaan ohjelmaan, kannattaa työmäärän vähentämiseksi (ja ohjelman pitämiseksi standardi ANSI-C++:n mukaisena) tyytyä Kerho-ohjelman tyyliseen keskustelevaan kyselyyn. Kuitenkin aina kun mahdollista, kannattaa käyttäjälle ehdottaa jotakin oletusvastausta, jonka hän voi hyväksyä pelkällä RET-näppäimen painamisella.

Jos joku haluaa oikeata näytön käsittelyä, on kurssin aliohjelmakirjastossa pieni apukirjasto `console.c`, jota saa käyttää (koska se voidaan kääntää useimmille laiteympäristöille). Tämän kirjaston avulla voidaan kirjoittaa yksi-ikkunaisia sovelluksia. Joka tapauksessa käyttöliittymä kannattaa ohjelmointivaiheessa eristää muusta koodista, jotta käyttöliittymän vaihtaminen olisi helpompaa (vrt. `kerhopp` –malliohjelmat).

2. Käyttöohje tai suunnitelma (Ohjelmointi, luku 2)

Ohjaajan ehdottamat muutokset on tehty suunnitelmaan ja suunnitelma on puhtaaksikirjoitettu. Jatkossa ohjelmaa kehitetään tämän suunnitelman pohjalta.

Tietysti mikäli myöhemmin keksii selviä parannuksia, niin näitä voidaan suunnitelmaan tehdä. Parannukset on kuitenkin hyväksytettävä ohjaajalla.

3. Apualiohjelmia tai ohjelman menut (MENUT.05)

Koska käytännön ohjelmointia tehdään sekä TOP-DOWN että BOTTOM-UP periaatteella, on aluksi (kun ei vielä osata paljoa C-kieltä) mukava tehdä BOTTOM-UP strategialla joitakin selvästi ohjelmassa tarvittavia aliohjelmia sekä valmiin ohjelman runkona toimiva valikkosysteemi.

Toisaalta voidaan edetä myös TOP-DOWN periaatteella ja kirjoittaa ohjelmasta toimiva runko, jossa mikään alakohta ei varsinaisesti toimi. Myöhemmin tähän on mukava yksi kerrallaan lisätä toimintoja.

4. Luokat ja tietorakenteet (Ohjelmointi, luvut 11-12)

Ohjelman jatkokehittelyn kannalta on välttämätöntä kiinnittää ainakin jonkinlainen osa ohjelman tarvitsemista tietorakenteista. Tietorakenteista piirretään aluksi kuvat ja ne hyväksytetään ohjaajalla ja sitten kirjoitetaan tietorakenteita vastaavat C-määrittelyt.

Tietorakenteiden ei välttämättä tarvitse olla samanlaiset kuin Kerho-ohjelmassa, mutta useisiin harjoitustöihin tämä rakenne käy lähes sellaisenaan, tai pienillä lisäyksillä.

Samalla suunnitellaan ohjelman luokkajako. Luokkajako kirjoitetaan CRC-korteille, jotka näytetään ohjaajalle.

5. Tietorakenteiden runko (malli RUNKO.1, JAKO.15)

Seuraavaksi kirjoitetaan monisteen mallia vastaava runko ohjelmasta, jossa ei ole vielä likikään kaikki toiminnot oikeasti toimivina, mutta tietorakenteen ja luokkajaon toimivuus voidaan testata.

Tässä seuraava vaihe ennen vaiheeseen 6 siirtymistä on kommentoida aliohjelmat ja jakaa yhdessä tiedostossa oleva koodi loogisiin kokonaisuuksiin (moduleihin). Tämän saa (ja kannattaa) tietysti tehdä valmiiksikin vaihetta 5 varten.

Viimeistään tässä vaiheessa (ja myöhemmissäkin) ohjaajat tenttaavat suullisesti tekijän ymmärtävän jokaisen ohjelmaansa kirjoittaman lauseen merkityksen (oma listaus saa olla mukana ja siinä saa olla kommentteja).

6. Tiedoston käsittely ja moduleihin jako (malli TALLETUS.2)

Ohjelmat suunnitellaan periaatteella:

```
ALKU
lue tiedostosta entiset tiedot
käsittele tietoa
talleta muuttuneet tiedot
LOPPU
```

Koska ohjelman runko on valmiina, ja ohjelman muita kohtia testattaessa täytyy aina syöttää riittävästi aineistoa toimintojen toteamiseksi, saattaa olla helpointa ottaa vanha aineisto tiedostosta ja tallettaa se lopuksi tiedostoon.

Siis seuraavaksi runkoon täydennetään tiedoston luku ja talletus. Ohjelmahan on jo jaettu modulehin, jotta jatkokirjoittaminen helpottuu. Tiedoston käsittelykin kannattaa lisätä kahdessa vaiheessa. Esimerkiksi ensin tiedoston kirjoittaminen ja kun tämä on testattu valmiiksi, niin lisätään tiedoston lukeminen.

7. Toimiva ohjelma (malli LUKEMINEN.3-ETSILAJ.5)

Nyt ohjelmaan täydennetään alkuperäisen suunnitelman mukaisia toimintoja pala kerrallaan koko ajan testaten. Näin saadaan lopuksi aikaan valmis ja toimiva ohjelma.

Harjoitustyössä ei välttämättä kaikkien alavalintojen tarvitse toimia, kunhan opiskelija on käynyt kaikki oleelliset ohjelmointiin ja C/C++ –kieleen liittyvät asiat läpi. Ohjaajan kanssa sovitaan mitkä kohdat ohjelmasta voidaan jättää toteuttamatta.

Toivottavaa olisi, että vähintään jokin yksinkertainen etsiminen (haku) toimisi.

Pääosin harjoitustyön tulee tasoltaan vastata liitteenä jaettua kerhon jäsenrekisteriä ja sen tulee toimia alkuperäisen suunnitelman mukaisesti!

2.8 Kurssin suoritus loppukokeella

Harjoitustyön tulee olla valmis ja hyväksytty ennen tenttiin osallistumista!

2.9 Kirjoitustyyli ja kommentointi

Harjoitustyö kirjoitetaan ja kommentoidaan samaan tyyliin kuin monisteen loppuosassa esitetyt esimerkkiohjelmat. Erityisesti harjoitustyöstä tarkastetaan

- muuttujien nimien käyttö ja kuvaavuus
- kommenttien riittävyys
- mikäli käytetään sellaisenaan monisteen esimerkkiohjelmia, on se ok. Mikäli niitä muutetaan, on kommentoitava mitä on muutettu.
- sisennysten loogisuus (2 välilyöntiä/sisennys riittää)
- vakioiden käyttö, eli ohjelmakoodissa ei saa esiintyä hämäräperäisiä numeerisia tai tekstivakioita, joita joskus mahdollisesti voisi joutua muuttamaan
- osoittimien oikea käyttö
- include-nimissä ei absoluuttisia polkuja, ei siis koskaan
  `#include "a:oma.h"`
  vaan
  `#include "oma.h"`
- ohjelma kääntyy täysin ilman varoituksia tai virheitä

- tietojen syöttö toimii
- ohjelmassa on vähintään yksi oikeellisuustarkistus
- ohjelmassa on jonkinlainen avustus
- ohjelmassa on toteutettu vähintään yksi tietorakenteen läpikäynti (“etsiminen”)

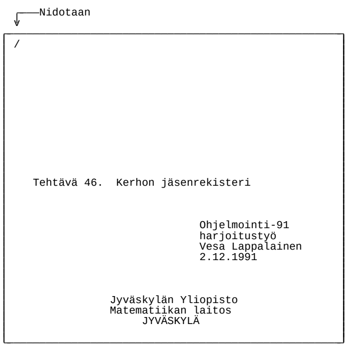
2.10 Työn palauttaminen

2.10.1 Ei selostusta

Työstä ei tarvitse kirjoittaa työselostusta, vaan hyvin kommentoitu ohjelmalistaus riittää.

2.10.2 Kansilehti

Ohjelmalistaus varustetaan kuitenkin seuraavanlaisella kansilehdellä, joka voi olla kirjoitettu joko koneella tai käsin (siististi!):



2.10.3 Ohjelman jako osiin

Ohjelma tulee olla jaettu osiin (projektiksi) vastaavasti kuin mallina olleessa kerho-ohjelmassa.

2.10.4 Ohjelmalistaus

Ohjelmalistaus tulostetaan niiltä osin, missä on tekijän omia muutoksia. Esim. kirjastoa `mjonot.c` (ja `mjonot.h`) EI SAA tulostaa. Tulostaminen tehdään esim. mikroluokasta komennolla

```
tulosta kerhomai.cpp kerho.h kerho.cpp
```

Tulostuskomento voi vaihdellakin, ks. luokan seinällä olevia ohjeita.

2.10.5 Alkuperäinen ohjelman suunnitelma

Ohjelman alkuperäinen suunnitelma tulee olla liitettynä ohjelman kommentteihin tai ohjelmalistauksen liitteeksi. Ohjelman toiminnan tulee vastata alkuperäistä suunnitelmaa. Tietysti tiettyjä parannuksia saa tehdä, mutta muuten kysymysten ja ulkoasun tulee vastata ohjelman toimintaa!

2.10.6 Toimiva ohjelma levykkeellä (ja OHJ.SIS)

Palautustilaisuuteen tulee tulla toimivan ja täysin virheettömästi kääntyvän ohjelman kanssa. Itse ohjelma ja sen tarvitsemat osat tulee olla helppo siirtää ohjaajan koneen levylle. Tämän voi tehdä esim. seuraavasti:

Kirjoita levykkeen päähakemistoon seuraavasta matkittu tiedosto `OHJ.SIS`:

```
\KURSSIT\CPP\KERHO\KERHO*.CPP
\KURSSIT\CPP\KERHO\KERHO*.H
\KURSSIT\CPP\KERHO\JASEN.CPP
\KURSSIT\CPP\KERHO\JASEN.H
\KURSSIT\CPP\KERHO\NAYTTO.CPP
\KURSSIT\CPP\KERHO\NAYTTO.H
\KURSSIT\CPP\KERHO\KELMIT.DAT
```

Eli yksinkertaisesti tiedosto, jossa sanotaan mitä tiedostoja (ja mistä hakemistosta) ohjaajan pitää kopioida korpulta omalle kovalevylleen harjoitustyön katsomiseksi.

Erityisesti pitää huomata, ettei projektitiedostoa saa kopioida, samoin kuin ei `.EXE` tiedostoa.

Tekijä kertoo ohjaajalle miten tehdään projekti, jolla ohjelma saadaan käännettyä (yleensä otetaan kaikki `.C` tiedostot mukaan, EI KOSKAAN `.H` tiedostoja).

2.10.7 Itsetehdyt aliohjelmat

Vaikka moni pystyykin tekemään harjoitustyönsä pienillä muutoksilla valmiista mallista, pitää jokaisen pystyä osoittamaan vähintään muutama TÄYSIN ITSE tehty ja suunniteltu aliohjelma TAI sitten hänen täytyy ymmärtää valmis malliohjelmakokonaisuus niin hyvin, että hän pystyy vastaamaan ohjaajan esittämiin malliohjelmaan liittyviin kysymyksiin!

2.10.8 Ohjelma on itse tehty (tai muuteltu)

Tekijän täytyy osoittaa ohjelmaa kohtaan niin suurta ymmärrystä, että ohjaaja varmistuu siitä, että tekijä on todella ohjelman tekijä! Ymmärrys tarkistetaan suullisilla kysymyksillä.

Vilpistä kiinni jääneille annetaan uusi harjoitustyön aihe!

2.10.9 EI SAA KÄYTTÄÄ strcpy yms. ilman HYVIÄ PERUSTELUITA

`strcpy`, `strcat`, `gets` yms. vaarallisten funktioiden käyttö on kielletty ilman hyviä perusteita siitä, miksi niitä voidaan käyttää!

2.10.10 Ohjelma kääntyy ilman varoituksia ja virheilmoituksia

Palautettavan ohjelman tulee kääntyä ilman mitään varoituksia ja virheilmoituksia. Tämä voidaan varmistaa esim. Borland C:ssä siten, että kohdassa `Options Compiler Messages` käydään laittamassa kaikki mahdolliset varoitukset ja virheilmoitukset päälle. Sitten kohdassa `Options Make` laitetaan `Break Make On Warnings` päälle.

2.10.11 Ohjelman toimivuus ei ole tae sen oikeellisuudesta!

Vaikka ohjelma toimisikin, ei tämä ole tae siitä, että ohjelma on virheetön. Erityisesti on muistettava testata kaikki erikoistapaukset kuten esimerkiksi:

- kerhoa ei ole
- lisätään 1. jäsen
- muistitila loppuu kesken (määrittele ohjelmaan jokin julman iso tilapäinen käyttämätön muuttuja, esim. taulukko)
- poistetaan jäsen
- poistetaan yhden jäsenen kerhon ainokainen jäsen
- jne…

Jopa –91 ja –92 malliohjelmissa (VAPAATYY.8) oli koneen kaatava vika kun poistettiin jäsen!

2.10.12 Tarkista osoittimet vielä kerran

On mahdollista, että ohjelma toimii tuurilla, vaikka siinä on jopa täysin vääriäkin lauseita. Eräs yleisimpiä virheitä on seuraava:

```
void ali(void)
{
  char *apu; /* Lokaali osoitin, jota ei muisteta alustaa */
  strcpy(apu,"Kissa");  /* VÄÄRIN! */
  /* ei edes kopioi_jono(N_S(apu),"Kissa") auta tilannetta */
  printf(apu);
}
```

Siis yleensä aloittelijalla kaikki lokaalit osoittimet ovat VAARALLISIA!

2.11 Työskentely mikroilla

2.11.1 Työskentely aina kovalevyllä

Mikroilla työskennellään siten, että käsiteltävät tiedostot ovat aina kovalevyllä! Miksi?

1. Siksi, että käännöksessä tarvitaan paljon levyä ja kovalevyn käyttö on huomattavasti nopeampaa kuin levykkeen käyttö.
2. Lisäksi levykkeen lukemisessa kosketetaan fyysisesti levykkeen pintaa ja levykkeen eliniästä tulee näin varsin rajallinen! Esimerkiksi edellisilläkin kurssilla joiltakin tuhoutui koko harjoitustyö, kun koko ajan käytettiin pelkkää levykettä (eikä ollut varmuuskopiota)!
3. Jos levy tuhoutuu, voidaan se tosin yrittää pelastaa, mutta pelastus**yritys** (joka voi olla turhakin) saattaa kestää 0.5-2 tuntia (VL), joten kannattaa miettiä saako samassa ajassa kirjoitettua kaiken uudelleen.
4. Lopuksi todettakoon, että virusten tartuntariski kasvaa sen mukaan mitä kauemmin levyke on koneessa.

2.11.2 Tee oma hakemisto

Kun tulet mikrolle, tee itsellesi aluksi oma alihakemisto sopivaan paikkaan (esimerkiksi `\OMAT\OHJELMOI\VESA`), ja siirry sinne.

2.11.3 Kopioi harjoitustyö levykkeeltä

Kirjoita aikaisemman mallin mukainen tiedostolista `OHJ.SIS` levykkeen päähakemistoon.

Käytä `HT.BAT` nimistä ajojonoa työn kopioimiseksi. Jos haluat kopioida ajojonon omaan kotikoneeseesi, on sinun kopioitava tiedostot `N:\BAT\HT.BAT` ja `N:\BIN\KOPIOI.EXE`. Ajojono `OHJHT.BAT` ks. alla tarvitsee lisäksi tiedoston `N:\BIN\CDD.COM`. Kaikki em. tiedostot on tehnyt VL, joten ne saa vapaasti kopioida.

Jatkossa aina omaan hakemistoon siirtymisen jälkeen voit kopioida levykkeeltä harjoitustyön esim. komennolla

```
C:\OMAT\OHJELMOI\VESA>ht ohj[RET]
```

Tämän jälkeen ota levyke pois koneesta!

2.11.4 Helpompi hakemiston luonti ja kopiointi

Vielä helpommaksi saat tietysti hommat jos kirjoitat itsellesi seuraavan ajojonon (`A:\OHJHT.BAT`):

```
@ECHO OFF
REM Luodaan oma ht-hakemisto ja otetaan ht sinne
CDD \OMAT\OHJELMOI\VESA /b /w
IF NOT errorlevel 1 HT OHJ
```

Nyt kun tulet koneelle voit aina kirjoittaa

```
C:\OMAT>a:ohjht[RET]
```

ja sitten poistaa levykkeen.

2.11.5 Käynnistä Borland-C

Borland-C:n saat käyntiin komennolla

```
C:\OMAT\OHJELMOI\VESA>bc[RET]
```

2.11.6 Ohjelmoi

Kirjoittele, kääntele, debuggaa ja kokeile.

2.11.7 Talleta pois lähtiessäsi muutokset levykkeelle

Kun lopetat työskentelysi, niin laita levyke takaisin koneeseen ja kirjoita komento

```
C:\OMAT\OHJELMOI\VESA>ht ohj[RET]
```

Näin kaikki työskentelyn aikana tapahtuneet muutokset tallettuvat takaisin levykkeelle, eikä levykettä ole käytetty liikaa, joten sille voi povata elinikää jopa koko kurssin ajaksi.

2.11.8 Ota varmuuskopio

Aina tietyin väliajoin kopioi koko levykkeesi tai ainakin sen tärkeät tiedostot toiselle levykkeelle. Näin sinulla on onnettomuuden sattuessa varakopio työstäsi.

Jos ensin yhden kerran kopioidaan tiedosto `A:\OHJ.SIS` varmuuskopiolevykkeelle, niin tämän jälkeen varmuuskopiotkin voidaan aina ottaa kovalevyltä kutsulla

```
C:\OMAT\OHJELMOI\VESA>ht ohj[RET]
```

Jos sinulle riittää, että korpulle tulee tasan vastaava hakemisto kuin kovalevylläkin on (siis esim. `A:\OMAT\OHJELMOI\VESA`), niin voit kirjoittaa pelkästään

```
C:\OMAT\OHJELMOI\VESA>ht[RET]
```

2.12 Kertaus mikrolla työskentelystä

Siis vielä lyhyesti joko (ilman `OHJHT.BAT`ia):

1. Kirjoita

```
C:\OMAT\>cd ohjelmoi\vesa[RET]
C:\OMAT\OHJELMOI\VESA>ht ohj[RET]
```

2. Korppu pois koneesta
3. Käynnistä Borland-C
4. Ohjelmoi
5. Poistu Borland-C:stä (***[Alt-X]***)
6. Korppu koneeseen
7. Kirjoita

```
C:\OMAT\OHJELMOI\VESA>ht ohj[RET]
```

tai `OHJHT.BATin` kanssa:

1. Kirjoita

```
C:\OMAT\>a:\ohjht[RET]
```

2. Korppu pois koneesta
3. Käynnistä Borland-C
4. Ohjelmoi
5. Poistu Borland-C:stä (***[Alt-X]***)
6. Korppu koneeseen
7. Kirjoita

```
C:\OMAT\OHJELMOI\VESA>a:\ohjht[RET]
```

3. Pääteohjaus 1

Tämän pääteohjauksen tarkoituksena on tutustua Ohjelmointi –kurssin malliohjelmina olevien KERHO ja TULKKI –ohjelmien toimintaan. Näistä voi saada joitakin ideoita oman harjoitustyön suunnitelmaan.

Seuraavat harjoitukset on parasta tehdä DOSissa. Siis jos olet Windowissa, napauta 2 kertaa DOS –ikonia päästäksesi DOS-tilaan.

Aivan ensimmäiseksi tee kovalevylle (esim. C) oma hakemisto (`C:\OMAT\OHJELMOI\nimi`, missä nimi on oma etunimesi).

3.1 Hakemistot

3.1.1 Kerho

KERHO –ohjelma tullaan kurssin aikana toteuttamaan TOP-DOWN suunnittelua käyttäen ja kehityksen eri vaiheita on seuraavissa hakemistoissa:

```
 Directory of N:\KURSSIT\CPP\KERHO
.            <DIR>      1.12.95   11.04
..           <DIR>      1.12.95   11.04
PAAMENU  0   <DIR>      1.12.95   11.23
MENUT    05  <DIR>      4.12.95   16.11
RUNKO    1   <DIR>     30.12.95   11.04
JAKO     15  <DIR>     30.12.95   11.04
TALLETUS 2   <DIR>     30.12.95   11.04
LUKEMINE 3   <DIR>     30.12.95   11.04
TARKISTU 4   <DIR>     30.12.95   11.04
ETSILAJ  5   <DIR>     30.12.95   11.04
VAINRAK  6   <DIR>     30.12.95   11.04
KENTTATA 7   <DIR>     30.12.95   11.04
VAPAATYY 8   <DIR>     30.12.95   11.04
```

Hakemiston nimessä oleva tarkennin 0-8 viittaa kehitysvaiheeseen (ei samat kuin harjoitustyön vaihenumerot). Kuhunkin alihakemistoon on käännetty valmiiksi ohjelma nimeltä `KERHO.EXE`. Tarkoituksena on katsoa eri vaiheissa olevan ohjelman toimintaa.

3.1.2 Tulkki

Tulkki ohjelma on toteutettu 3 vaiheessa.

```
 Directory of N:\KURSSIT\C\TULKKI
.            <DIR>     30.12.91   11.05
..           <DIR>     30.12.91   11.05
VALITES  1   <DIR>     30.12.91   11.05
SANASLUK 2   <DIR>     30.12.91   11.05
VALMIS   3   <DIR>     30.12.91   11.05
```

Eri vaiheiden testaaminen vaatii debuggerin käytön osaamista, ja tässä pääteohjauksessa tutustutaan vain viimeiseen vaiheeseen eli alihakemistoon `VALMIS.3`, josta löytyy "valmis" tulkki-ohjelma nimeltä `TULKKI.EXE`.

3.2 Kerho

3.2.1 Päämenu

Kerho ohjelman ensimmäisessä vaiheessa on tehty ohjelma, joka tulostaa vain pelkän ohjelman päävalikon.

1. Siirry alihakemistoon `N:\KURSSIT\CPP\KERHO\PAAMENU.0`
2. Käynnistä ohjelma kirjoittamalla

```
N:\KURSSIT\CPP\KERHO\PAAMENU.0>KERHO[RET]
```

3.2.2 Päämenu

Seuraavassa vaiheessa on toimivat valikot, mutta mikään alakohta ei toimi.

1. Siirry alihakemistoon `N:\KURSSIT\CPP\KERHO\MENUT.05`
2. Käynnistä ohjelma kirjoittamalla

```
N:\KURSSIT\CPP\KERHO\MENUT.05>KERHO[RET]
```

3.2.3 Runko

Seuraava vaihe on lisätä ohjelmaan niin paljon koodia, että tietorakenteet saadaan toimiviksi. Mitään järkevää ei vielä saada aikaan, mutta rakenteiden toimivuus voidaan todeta.

1. Siirry alihakemistoon `N:\KURSSIT\CPP\KERHO\RUNKO.1`
2. Käynnistä KERHO-ohjelma
3. Kokeile tulostusta (***3***).
4. Lisää jäsen pari kertaa.
5. Kokeile tulostusta.
6. Kokeile muita toimintoja.
7. Lopeta ohjelma.
8. Käynnistä ohjelma uudelleen.
9. Kokeile tulostusta.
10. Lopeta ohjelma

3.2.4 Talletus

Seuraavana ohjelmaan lisätään talletus, jotta syöttöaineistoa on riittävästi käytössä myöhemmin, kun muita toimintoja lisätään. Ohjelma ei vieläkään osaa lukea päätteeltä, mutta se osaa lukea ja kirjoittaa tiedoston, jossa jäsenistö on.

1. Siirry alihakemistoon `N:\KURSSIT\CPP\KERHO\TALLETUS.2`
2. Siirry C-levyn omaan hakemistoosi (jollei ole, tee se).

3. Kopioi N-levyn *.DAT –tiedostot

```
C:\OMAT\OHJELMOI\VESA>copy n:*.dat
```

4. Käynnistä KERHO-ohjelma ***`n:kerho`***
5. Vastaa kerhon nimeksi ***`KELMIT`***
6. Kokeile tulostusta (***`3`***).
7. Lisää jäsen.
8. Kokeile tulostusta.
9. Kokeile muita toimintoja.
10. Lopeta ohjelma.
11. Käynnistä ohjelma uudelleen.
12. Kokeile tulostusta. Onko lisätty jäsen tallessa?
13. Lopeta ohjelma.
14. Käynnistä vaikkapa Borland-C editori ja editoi sillä tiedostoa `KELMIT.DAT`:

    ```
    C:\OMAT\OHJELMOI\VESA>BC KELMIT.DAT[RET]
    ```

    Lisää tiedostoon jokin uusi jäsen ja poista pari muuta.
15. Lopeta Borland-C (***`[Alt-X]`***).
16. Käynnistä KERHO-ohjelma jälleen kerran.
17. Kokeile tulostusta. Onko muutokset kunnossa?
18. Lopeta ohjelma.

3.2.5 Päätesyöttö

Kun tiedot osataan tallettaa tiedostoon, voidaankin ohjelmaan jo lisätä päätesyöttö. Tässä vaiheessa ei vielä huolehdita oikeellisuustarkistuksista.

1. Siirry alihakemistoon `N:\KURSSIT\CPP\KERHO\LUKEMINE.3`
2. Siirry omaan hakemistoosi ja kopioi jälleen `N:*.DAT`
3. Käynnistä KERHO-ohjelma
4. Vastaa kerhon nimeksi ***`KELMIT`***
5. Kokeile tulostusta (***`3`***).
6. Lisää jäsen.
7. Kokeile tulostusta.
8. Lisää toinen jäsen, ja vastaa kysymyksiin mielettömiä arvoja. Kelpaavatko?
9. Kokeile tulostusta.
10. Lopeta ohjelma.

3.2.6 Oikeellisuustarkistukset

Kun päätesyöttö toimii luotettavasti, voidaan lisätä syöttöön oikeellisuustarkistukset.

1. Siirry alihakemistoon `N:\KURSSIT\CPP\KERHO\TARKISTU.4`
2. Siirry omaan hakemistoosi ja kopioi jälleen `N:*.DAT` sekä `N:*.HLP`
3. Käynnistä KERHO-ohjelma

4. Vastaa kerhon nimeksi ***KELMIT***
5. Kokeile tulostusta (***3***).
6. Lisää jäsen.
7. Kokeile tulostusta.
8. Lisää toinen jäsen, ja vastaa kysymyksiin mielettömiä arvoja. Kelpaavatko?
9. Kokeile tulostusta.
10. Lopeta ohjelma.

3.2.7 Etsiminen ja korjailu

Seuraavaksi lisätään vielä ohjelmasta puuttuvia toimintoja. Esimerkiksi tietojen etsiminen ja korjailu.

1. Siirry alihakemistoon `N:\KURSSIT\CPP\KERHO\ETSILAJ.5`
2. Siirry omaan hakemistoosi ja kopioi jälleen `N:*.DAT`, `N:*.HLP`
3. Käynnistä KERHO-ohjelma
4. Vastaa kerhon nimeksi ***KELMIT***
5. Kokeile tulostusta (***3***).
6. Kokeile tietojen etsimistä (***2***).
7. Valitse kysymysmerkki (***?***) jotta näet mitä kenttiä on käytössä.
8. Valitse vaikkapa nimikenttä (***1***).
9. Etsi kaikki A:lla alkavat nimet (***a****).
10. Valitse vaikkapa nimikenttä (***1***).
11. Etsi kaikki nimet jotka ovat aakkosissa ennen nimeä KISSA (***<kissa***).
12. Selaile löytyneitä nimiä (**+** ja **-**).
13. Palaa päävalintaan (***[Ret]***).
14. Koeta muuttaa James Bondin tietoja.
15. Kokeile etsiä kaikki joilla ei ole työpuhelinta (hakuehdoksi puhelinkenttään **==**).
16. Lopeta ohjelma.

3.2.8 Yleistystä

Seuraavat ohjelmankehitysvaiheet liittyvät rakenteen yleistämiseen, eivätkä tarjoa mitään ulospäin näkyviä toiminnallisia muutoksia. Viimeinen kehitysvaihe ylittää jo ohjelmointikurssin vaatimustason; siinä yleistys on viety niin pitkälle, että rekisterin tietueiden tiedot on talletettu `.RAK` –nimisiin tiedostoihin. Näin jokainen voi periaatteessa kokeilla voiko hän saada oman harjoitustyönsä aikaiseksi muuttamalla tiedostoa `OLETUS.RAK`.

1. Siirry alihakemistoon `N:\KURSSIT\C\KERHO\VAPAATYY.8`
2. Siirry omaan hakemistoosi ja kopioi nyt `N:*.DAT`, `N:*.HLP` ja `N:*.RAK`
3. Käynnistä KERHO-ohjelma.
4. Vastaa kerhon nimeksi ***KELMIT***
5. Kokeile avustusta (***?***).

6. Kokeile tulostusta (***3***).
7. Laita kaikkien henkilöiden maksettu maksu kentään 0 mk.
   (Voit kokeilla avustusta ? missä kohti vaan).

```
Muutetaan muuttamisessa käytettävät kentät.
Kentät >B[RET]
Maksettu maksu mk >0.0[RET]
2 = täytä kentät..
Valitse kenttä... :1
Jäsenen nimi ...>*
```

8. Kokeile muuttaa jäsenmaksuksi 10 mk kaikille niille, jotka ovat liittyneet ennen vuotta 1986.
9. Lopeta ohjelma.
10. Käynnistä KERHO-ohjelma.
11. Vastaa kerhon nimeksi ***KOULU***
12. Kokeile tulostusta (***3***).
13. Kokeile lisätä koululaisia.
14. Lopeta ohjelma.
15. Kopioi `KOULU.RAK` nimelle `OLETUS.RAK`

```
C:\OMAT\OHJELMOI\VESA>COPY KOULU.RAK OLETUS.RAK[RET]
```

16. Muokkaa tiedostoa `OLETUS.RAK` vastaamaan osittain omaa harjoitustyötäsi. Ohjeita rakennetiedoston kirjoittamiseen löydät tiedostoston `KERHOLRA.C` alkukommenteista.

```
C:\OMAT\OHJELMOI\VESA>BC OLETUS.RAK[RET]
```

17. Lopeta Borland-C (***[Alt-X]***).
18. Käynnistä KERHO-ohjelma.
19. Vastaa tiedoston nimeksi ***OMA*** ja kokeile toimiiko.

Harjoitustyöksi ei kuitenkaan kelpaa tämän hakemiston tiedostojen listaaminen!

3.3 Tulkki

Tulkki-ohjelmasta kokeilemme vain “valmista” ohjelmaa.

1. Siirry hakemistoon `N:\KURSSIT\C\TULKKI\VALMIS.3`
2. Sanasto on talletettu tiedostoon `SANASTO.DAT`. Katso mitä sanoja on sanastossa.
3. Käynnistä TULKKI-ohjelma

```
N:\KURSSIT\C\TULKKI\VALMIS.3>TULKKI[RET]
```

4. Paina kysymysmerkkiä ja return-näppäintä (***?[RET]***).
   Kokeile tulkkaamista vaikkapa seuraaville sanoille

```
HÄN suomi -> englanti
HE  englanti -> ruotsi
```

5. Kokeile muitakin käännöksiä (muista, että myös jokerimerkki * toimii, ja että sanaston koko on rajallinen!).

6. Lopeta ohjelma (***`[Ret]`***).
7. Käynnistä ohjelma jollakin sanalla:

```
N:\KURSSIT\C\TULKKI\VALMIS.3>TULKKI sukka[RET]
```

8. Lopeta ohjelma.
9. Kopioi sanasto omaan hakemistoosi ja lisää sanastoon savon-kieli ja kokeile toimintaa. Iso kirjain tarkoittaa kielen valintaa käännöksessä. Miten siis lisäät savon kielen?

3.4 C-kielen opetusohjelma

C-kielen opetusohjelma on matematiikan laitoksella erikoistyönä tehty harjoitelma "interaktiivisesta" C-kielen opetusohjelmasta. Ohjelman voit kopioida itsellesi omaan koneeseesi kopioimalla levykkeelle hakemistot:

```
I:\C_OPETUS\DISK1
I:\C_OPETUS\DISK2
```

3.4.1 Ensimmäinen ohjelma

Tutkitaan aluksi mitä ohjelma osaa sanoa yksinkertaisesta C-ohjelmasta:

1. Etsi Windowsin ikoneista ryhmä `C-kielen opetus`
2. Klikkaa opetusohjelma käyntiin.
3. Valitse ylärivin työkaluista sisällysluettelo
4. Valitse sisällysluettelosta `Ensimmäinen ohjelma`
5. Kulje hiirellä ohjelma eri komponentteja ja täydennä luentomonisteen luvun 7 ensimmäiseen malliohjelmaan itsellesi tärkeimmät asiat.
6. Valitse seuraavaksi sisällysluettelosta `Ohjelman synty` ja tutki miten tekstitiedosto muuttuu ajettavaksi ohjelmaksi

3.4.2 Miten ohjelmalauseita "keksitään"

Seuraavaksi tutkimme miten "mystisiltä" näyttäviä ohjelmarakenteita voidaan "keksiä":

1. Valitse sisällysluettelosta `Ohjelman kehityksestä`
2. Aja animaatio asiaan keskittyen läpi. Palaa tähän animaatioon kurssin myöhemmissä vaiheissa jos asiat tuntuvat vaikeilta.
3. Taulukoista löytyy mukava esimerkki kohdasta `Merkkijonot`

3.4.3 Opetusohjelman HTML-versio

Katso myös vastaavaa HTML-versiota osoitteesta:

```
http::/www.math.jyu.fi/__________
```

3.4.4 Kurssin WWW-sivut

Katso kurssin WWW-sivuja osoitteesta

http::/www.math.jyu.fi/~vesal/ohjelmointi98

4. Pääteohjaus 2

Seuraavien harjoitusten tarkoituksena on opettaa ohjelmointiympäristön käyttöä. Itse C-kielisten ohjelmien syntaksiin ei vielä ole syytä takertua liiaksi.

Tässä harjoituksessa opittavat asiat tulevat hyödyksi ohjelman koodaamisessa ja testaamisessa. Itse ohjelman suunnittelun kanssa tällä harjoituksella ei ole mitään tekemistä!

4.1 Ohjelman kirjoittaminen ja kääntäminen

Jos olet Windowsissa, on tämä harjoitus vielä parasta tehdä DOSista käsin. Siis aukaise yksi DOS-ikkuna 2-näpäyttämällä DOS-ikonia.

Siirry omaan hakemistoosi (`C:\OMAT\OHJELMOI\nimi`). Jos hakemistoa ei ole, luo hakemisto.

Katso `DIR`-komennolla mitä tiedostoja on hakemistossa.

1. Kirjoita Borland C:llä seuraava C-ohjelma nimelle `JARJ2.C` (käynnistyy komennolla ***`BC jarj2.c[RET]`***). Yritä saada vastaavat sisennykset ja ole tarkkana kirjainten `1` (yksi), `i` (ii, india) ja `l` (ällä, lima) suhteen. Myös isojen ja pienten kirjainten kanssa on syytä olla tarkkana! Jos huomaat mallissa kirjoitusvirheitä, niin älä korjaa niitä nyt, koska tarkoituksena on opetella tulkitsemaan kääntäjän virheilmoituksia.

```
/* Ohjelma järjestää kaksi lukua suuruusjärjestykseen. */
#include <stdio.h>
int main(void)
{
  int x1,x2,t;
  printf("Anna 1. luku >’); scanf("%d",&x1);
  printf("Anna 2. luku >"); scanf("%d",&x2);
  if ( x1 > x2 ) {
    t  = x1;
    x1 = x2;
    x2 = T;
  }
  printf("Luvut: %d %d.\n",x1,x2)
  return 0;
}
```

2. Mitä luulet ohjelman tulostavan?
3. Talleta valmis ohjelma (***`[F2]`***).
4. Käännä ohjelma (***`[Alt-C]M`*** tai ***`[Alt-F9]`***).
5. Mikäli sait ohjelman kirjoitettu aivan vastaavasti kuin mallissa, ilmestyy ruudun alalaitaan “MESSAGE”-ikkuna, johon tulee tekstit:

```
Compiling JARJ2.C:
Error JARJ2.C 7: Undefined symbol 'd'
Error JARJ2.C 7: Function call missing )
Error JARJ2.C 7: Unterminated string or character constant
Warning JARJ2.C 9: Possible use of 'x1' before definition
Error JARJ2.C 12: Undefined symbol 'T'
Error JARJ2.C 15: Statement missing ;
Warning JARJ2.C 16: Function should return a value
Warning JARJ2.C 16: 't' is assigned a value that is never used
```

6. Liikuttamalla kursoria ylös ja alaspäin virheikkunassa, nähdään virheiden paikat (likimäärin!) ohjelmakoodissa. Kokeile!
7. Siirry ensimmäisen virheen kohdalle ja paina ***`[RET]`***.
8. Huomaat varmaakin, että värit osoittavat jotakin ohjelmakoodissa. Mitä? Värit auttavat paikallistamaan useita virheitä.
9. Korjaa virhe! Virhe ei välttämättä ole kursorin kohdalla, vaan se vasta paljastuu siinä kohdassa. Sama virhe saattaa poikia useita eri virheilmoituksia!
10. Paina ***`[Alt-F8]`*** kunnes olet seuraavassa todellisessa virheessä. Korjaa ja jatka kunnes kaikki virheet on korjattu.
11. Käännä ohjelma uudelleen.
12. Talleta korjattu ohjelma.
13. Aja ohjelma (***`[Alt-R]R`***). Ohjelman tulostuksen saat uudelleen näkyviin painamalla ***`[Alt-F5]`*** tai voit avata uuden ikkunan tulostusta varten (***`[Alt-W]O`*** ja siirrä ikkunan hiirellä sopivaksi).

4.2 Borland C-editorin käyttöohjeet

4.2.1 Ikkunat

Borland-C ympäristö on suurinpiirtein seuraavan näköinen:

```
  =  File  Edit  Search  Run  Compile  Debug  Project  Options    Window  Help
┌─[þ]──────────────────────┌────────────────────────┐──────────────────────1─[ ]─┐
│/* Ohjelma järjestää ka │ Run             Ctrl-F9 │ seen. */
│#include <stdio.h>      │ Program reset  Ctrl-F2 │
│                        │ Go to cursor        F4 │
│int main(void)          │ Trace into          F7 │
│{                       │ Step over           F8 │
│  int x1,x2,t;          │ Arguments...           │
│  printf("Anna 1. luku  └────────────────────────┘
│  printf("Anna 2. luku >"); scanf("%d",&x2);
│  if ( x1>x2 ) {
│    t  = x1;
│    x1 = x2;
│    x2 = T;
│  }
│  printf("Luvut: %d %d.\n",x1,x2)
│  return 0;
│}
│
│
```

1. Näyttö koostuu siis useista ikkunoista. Käytössä olevan ikkunan ympärillä on kaksinkertainen raami. Ikkunoiden kokoa ja paikkaa voidaan muuttaa hiirellä tarttumalla kiinni ikkunan yläreunaan tai ikkunan johonkin nurkkaan. Kokeile!
2. Ikkunasta toiseen voidaan siirtyä joko hiirellä näpäyttämällä sitä ikkunaa, johon halutaan siirtyä tai painamalla ***`[Alt-?]`***, missä ? on sen ikkunan numero (oikeassa yläkulmassa ikkunaa) johon halutaan siirtyä. Kokeile!

4.2.2 Menut

3. Toimintoja voidaan valita joko hiirellä yläreunasta, tai painamalla ***[Menu]*** (***[F10]***) ja siirtymällä kursorilla halutun toiminnon kohdalle. Toiminto voidaan valita myös painamalla toiminnon korostetua kirjainta. Siis vaihtoehtoja on useita. Alkuksi menujen käyttö on näppärää nuolinäppäimillä tai hiirellä, mutta haluttaessa tehokkaampaa toimintaa, kannattaa käyttää suoravalintoja.
4. Kokeile ajaa edellinen ohjelma neljällä eri tavalla:
   a. valitse hiirellä menuista **R**un ja sieltä **R**un.
   b. paina ***[F10]***, liiku kursorilla kohtaan **R**un. Paina ***[RET]***. Siirry nuolilla kohtaan **R**un. Paina ***[RET]***.
   c. paina ***[Alt-R]R***
   d. paina ***[Ctrl-F9]***
5. Mikä edellistä tavoista tuntuu parhaalta?
6. Mistä keksittiin ***[Ctrl-F9]***?
7. Alla on lueteltu funktionäppäinten käyttötarkoitukset. Selaile menuista kullekin funktionäppäimelle myös menuvalinta (tyyliin `[Ctrl-F9] = [Alt-R]R`):

4.2.3 Funktio-näppäimet

```
[F1]           - apua
[F2]           - talleta tiedosto
[F3]           - avaa tiedosto
[F4]           - aja ohjelmaa kursoririville saakka
[F5]           - zoom (muuta ikkuna isoksi/pieneksi)
[F6]           - seuraava ikkuna
[F7]           - suorita seuraava rivi
[F8]           - suorita seuraava rivi käskynä
[F9]           - käännä ja linkitä ohjelma
[F10]          - menu
[Alt-F1]       - viimeksi katsottu apua-ruutu
[Alt-F3]       - sulje ikkuna
[Alt-F4]       - muuttujan arvon tutkiminen
[Alt-F5]       - vaihda tulostusruutuun/editoriin
[Alt-F7]       - siirry edelliseen virheeseen
[Alt-F8]       - siirry seuraavaan virheeseen
[Alt-F9]       - käännä ohjelma
[Ctrl-F1]      - apua kursorin kohdalla olevasta sanasta
[Ctrl-F2]      - alusta ohjelma
[Ctrl-F3]      - kutsupino
[Ctrl-F4]      - muuta muuttujan arvoa
[Ctrl-F5]      - siirrä ikkunan paikkaa/muuta ikkunan kokoa
[Ctrl-F6]      -
[Ctrl-F7]      - lisää muuttuja WATCH-ikkunaan
[Ctrl-F8]      - poista/lisää breakpoint
[Ctrl-F9]      - käännä ja linkitä tarvittaessa,  aja ohjelma
[Alt-0]        - valintalista ikkunoista
[Alt-#]        - siirry ikkunaan nro #
[Alt-W]W       - siirry WATCH-ikkunaan
[Alt-X]        - poistu käyttöjärjestelmään
[Ctrl-Ins]     - laita lohko leikekirjaan
[Shift-Ins]    - ota lohko leikekirjasta
[Shift-Del]    - poista lohko ja laita se leikekirjaan
[Ctrl-Del]     - poista lohko
[Ctrl-Y]       - tuhoa rivi
[Ctrl-T]       - poista sana
[Ctrl-nuolet]  - liiku sana kerrallaan
[Home]         - rivin alkuun
[End]          - rivin loppuun
[PgUp]         - näytöllinen ylöspäin
[PgDn]         - näytöllinen alaspäin
[Ctrl-PgUp]    - tiedoston alkuun
[Ctrl-PgDn]    - tiedoston loppuun
[Esc]          - poistu menusta tai apua-toiminnosta
[Alt-BS]       - UNDO, eli peru muutokset
[Ctrl-Q]]      - siirry sulun vastinpariin vasemmalle ( { [ /* "
[Ctrl-Q][      - siirry sulun vastinpariin oikealle ) } ] */ "
[Ctrl-Q]F      - etsi
[Ctrl-K]B      - aloita lohko (vasen ylänurkka)
```

```
[Ctrl-K]K      - lopeta lohko
[Shift-nuolet] - maalaa lohkoa
[Ctrl-K]W      - kirjoita lohko levylle
[Ctlr-K]R      - lue lohko levyltä
```

4.2.4 Ikkunan osat ja painikkeet

Usein jokin ikkuna saattaa olla jakautunut useisiin ali-ikkunoihin. Esimerkiksi seuraava ikkuna saadaan ***[Alt-O]EP***:

```
┌[*]────────────── Preferences ──────────────────┐
│                                                │
│ Screen Size            Auto Save               │
│  (.) 25 lines           [X] Editor files       │
│  ( ) 43/50 lines        [X] Environment        │
│                         [X] Desktop            │
│ Source Tracking         [X] Project            │
│  (.) New window                                │
│  ( ) Current window     [ ] Save old messages  │
│                                                │
│               OK        Cancel       Help      │
│                                                │
└────────────────────────────────────────────────┘
```

1. Nyt ali-ikkunoina on otsikoiden alla olevat tekstit (Screen Size, Auto Save, jne). Ali-ikkunasta toiseen voidaan siirtyä ***[Tab]*** ja ***[Shift-Tab]*** –näppäimillä. Kokeile!
2. Painikkeita on 2 eri tyyppiä: radionäppäimet (yhtä painamalla muut ponkaisevat ylös, vrt. vanha matkaradio) ja tavalliset painikkeet. Painike voidaan painaa hiirellä tai painamalla ***[Alt-kirjain]*** tai siirtymällä ko. ali-ikkunaan ja liikkumalla nuolilla ja välilyöntinäppäimellä. Kokeile!

4.2.5 Kääntäjän virheilmoitukset

Borlandin kääntäjissä voidaan monipuolisesti asetella optioita siitä, mistä kääntäjä valittaa. Yleisesti ottaen kannattaa pitää kaikki mahdolliset virheilmoitukset päällä, koska varoitus on käytännössä jotakin vikaa ohjelmassa.

Kaikki virheilmoitukset saat päälle seuraavasti:

1. Valitse: `Options Compiler Messages Display`
2. Katso, että `All` on valittu.
3. Valitse `Options Make`
4. Katso, että kohdassa `Break Make On` on kohta `Warnigs` –valittuna.

4.2.6 Help-toiminto

1. Borlandin ohjelmaympäristöissä on varsin hyvä Help-toiminto. ***[F1]***-näppäimellä saadaan tavallinen Help. Kokeile! Mikä on esimerkiksi komento lohkon sisentämiseksi?
2. ***[Ctrl-F1]*** antaa apua siitä sanasta, jonka kohdalla kursori on. Kokeile vaikkapa sanoihin `printf`, `scanf` ja `return`. Esimerkiksi `printf`:n kohdalla voit jatkaa helpissä `format`-kohtaan ja katsoa millainen `format` voi olla.
3. Tätä käyttäen voidaan yleensä aina löytää syntaksi halutulle käskylle. Myöhemmin tässä ohjauksessa käytetään satunnaislukugeneraattoria `rand`. Kokeile löytää kutsun syntaksi kirjoittamalla koodiin ylimääräinen `rand` ja paina

[Ctrl-F1]. Yleensä helpistä löytyy myös valmis esimerkki funktion tai käskyn käytöstä. Tämä esimerkki tai sen osa voidaan tarvittaessa kopioida omaan koodiin. Muista lopuksi pyyhkiä `rand` pois koodista.

4.2.7 Kääntäjän tekemät tiedostot

1. Poistu käyttöjärjestelmään ja kirjoita ylös mitä juuri kirjoittamaasi ohjelmaan liittyviä tiedostoja on syntynyt hakemistoosi hakemistoosi (joskus kääntäjä on asetettu siten, että käännöksen tulokset menevät RAM-levylle. Tarkista siis sekin) ja mitä varten mikäkin tiedosto on:

   ____
   ____
   ____
   ____
   ____
   ____

2. Anna käyttöjärjestelmässä komento ***JARJ2[RET]***. Mitä tapahtuu? Miksi?

4.3 Borland C debugger

Käynnistä Borland C uudelleen ja huolehdi siitä, että saat tiedoston `JARJ2.C` näkyviin.

1. Siirrä myös WATCH-ikkuna näkyviin (***[Alt-W]W***).
2. Laita **yksi kerrallaan** WATCH-ikkunaan muuttujat `x1`, `x2` ja `t` (***[Ctrl-F7]***). Watch ikkunaan saa muuttujia myös siirtymällä tekstissä muuttujan nimen päälle ja painamalla sitten ***[Ctrl-F7]***. Laita WATCH-ikkunaan myös ehdon `x1>x2` arvo (***[Ctrl-F7]x1>x2[RET]***). Aluksi muuttujien perässä voi olla virheilmoitus jonka pitäisi poistua kun ohjelman suoritus aloitetaan ja muuttujan "elämä alkaa".
3. Aja ohjelmaa rivi kerrallaan (***[F7]***).
4. Katso välillä tulostusruutua (***[Alt-F5]***).
5. Kokeile uudestaan. Käytä myös kursoriin saakka ajamista (eli siirrä kursori vaikkapa riville `x2=t` ja paina ***[F4]***).
6. Siirrä vaihtamisessa suoritettavat lauseet samalle riville:
   ```
   t=x1; x1=x2; x2=t;
   ```
   Eli:
   a. siirry rivin loppuun (***[End]***)
   b. poista rivin loppu (***[Ctrl-T]*** tai (***[Del]***)
   c. poista seuraava sana (***[Ctrl-T]***)
   d. lisää välilyönti
7. Kokeile suoritusta rivi kerrallaan uudelleen, kun ko. lauseet on siirretty.
8. Kokeile suoritusta rivi kerrallaan käyttämällä nyt näppäintä ***[F8]***. Mitä eroa näppäimeen ***[F7]*** ________________________.

9. Aja vielä kerran riville `if ( x1...` Muuta muuttujan `x1` arvoa (***[Ctrl-F4]x1[RET]***) ennenkuin jatkat suorittamista rivi kerrallaan.
10. Lisää ohjelmaan tulostusrivi, joka kertoo ohjelman käyttäjälle mitä ohjelma tekee.
11. Talleta ohjelma lopuksi (***[F2]***).

4.4 Lohkojen käsittely

1. Aloita uusi ohjelma nimeltä `JARJ3.C` (***[F3] jarj3.c[RET]***)
2. Lue pohjaksi edellinen ohjelma (***[Ctrl-K]R jarj2.c[RET]***).
3. Tarkoituksena on muuttaa ohjelmaa siten, että järjestetäänkin 3 lukua suuruusjärjestykseen (rivit on numeroitu jotta alla voidaan viitata riveihin; numeroita ei missään tapauksessa kirjoiteta!):
LUE ENSIN HUOLELLISESTI KOHTIEN 4 JA 5 OHJEET!

```
01  /* Ohjelma järjestää kolme lukua suuruusjärjestykseen.*/
02  #include <stdio.h>
03  int main(void)
04  {
05    int x1,x2,x3,t;
06    printf("Anna 1. luku >"); scanf("%d",&x1);
07    printf("Anna 2. luku >"); scanf("%d",&x2);
08    printf("Anna 3. luku >"); scanf("%d",&x3);
09    if ( x1>x2 ) {
10      t=x1; x1=x2; x2=t;
11    }
12    if ( x1>x3 ) {
13      t=x1; x1=x3; x3=t;
14    }
15    if ( x2>x3 ) {
16      t=x2; x2=x3; x3=t;
17    }
18    printf("Luvut: %d %d %d\n",x1,x2,x3);
19    return 0;
20  }
```

4. Kolmannen luvun lukeminen voidaan kopioida vaikkapa 2. luvun lukemisesta (rivinumerot viittaavat valmiiseen koodiin):
   a. siirry rivin 07 **alkuun** (***nuolet*** ja ***[Home]***)
   b. merkitse lohkon alku (paina ***[Shift]*** pohjaan ja pidä se siellä)
   c. siirry seuraavan rivin alkuun (***nuoli alas***)
   d. merkitse lohkon loppu (päästä ***[Shift]*** ylös)
   e. laita lohko leikekirjaan (***[Ctrl-Ins]***)
   f. ota lohko leikekirjasta (***[Shift-Ins]***, mahd uudestaan jos rivi ei tullut kahdesti)
   g. tee riville tarvittavat muutokset
5. Vertailut saadaan vastaavasti kopioimalla rivit 09-11 (ed. esim. mukaiset valmiit rivinumrot) kahteen kertaan:
   a. siirry rivin 09 alkuun (***nuolet*** ja ***[Home]***)
   b. merkitse lohkon alku (***[Shift]*** pohjaan ja pidä siellä)
   c. siirry rivin 12 alkuun (***nuolet alas***)
   d. merkitse lohkon loppu (***[Shift]*** ylös)
   e. laita lohko leikekirjaan (***[Ctrl-Ins]***)
   f. kopioi lohko (***[Shift-Ins]***, mahd. uudestaan jos vanha hävisi)

g. kopioi lohko (***[Shift-Ins]***)
h. tee riveille tarvittavat muutokset

6. Tee loput tarvittavat muutokset.
7. Talleta ohjelma.
8. Lisää myös `x3` WATCH-ikkunaan.
9. Ajele ohjelmaa ja tutki toimiiko.

Lohkon voi merkitä myös hiirellä. Kokeile sitäkin, mutta muista että näppäimistö on aina nopeampi ja tarkempi (kun sen kunnolla opettelee)!

4.5 Aliohjelmat

1. Muokkaa tiedostosta `JARJ3.C` seuraava tiedosto `JARJ3A.C`:

```
/* Ohjelma järjestää kolme lukua suuruusjärjestykseen.*/
#include <stdio.h>
/* Aliohjelma vaihtaa kaksi lukua keskenään. */
void vaihda(int *a, int *b)
{
  int t;
  t = *a; *a = *b; *b = t;
}
int main(void)
{
  int x1,x2,x3;
  printf("Anna 1. luku >"); scanf("%d",&x1);
  printf("Anna 2. luku >"); scanf("%d",&x2);
  printf("Anna 3. luku >"); scanf("%d",&x3);
  if ( x1 > x2 ) vaihda(&x1,&x2);
  if ( x1 > x3 ) vaihda(&x1,&x3);
  if ( x2 > x3 ) vaihda(&x2,&x3);
  printf("Luvut: %d %d %d\n",x1,x2,x3);
  return 0;
}
```

2. Laita WATCH-ikkunaan muuttujat x1, x2, x3, a, b ja t.
3. Kokeile ajaa askel kerrallaan käyttäen sekä ***[F7]*** että ***[F8]***. Mikä on näppäinten välinen ero? _______________________.
4. Poista aliohjelmasta kaikki tähdet (`*a` → `a` jne) sekä pääohjelmasta aliohjelman `vaihda` kutsussa olevat &-merkit.
5. Talleta muutettu ohjelma nimelle `JARJ3AA.C` (***[Alt-F]A[RET]jarj3aa.c[RET]***).
6. Kokeile muutetun ohjelman toimintaa askel kerrallaan. Muuttuvatko `x1, x2` ja `x3` aliohjelman suorituksen aikana kuten edellisellä kerralla?
7. Avaa edellinen `jarj3a.c` uudelleen ja kokeile vielä kerran suoritusta askel kerrallaan. Vertaa toimintaa ohjelmaan `jar3aa.c`.

4.6 C++

Sama ohjelma voidaan tehdä myös C++:lla. Erona C++:ssa on että syötössä ja tulostuksessa voidaan käyttää << ja >> –operaattoria (voidaan tulostaa ja lukea myös kuten C-kielessä) sekä voidaan käyttää referenssiä &, jonka ansiosta aliohjelmassa tehdyt muutokset heijastuvat pääohjelmaan vaikkei parametreja viedäkään osoittimien avulla.

Avaa `jarj3aa.c` ja lisää siihen aluksi vain muutos:

```
void vaihda(int &a, int &b)
```

Talleta nimelle `jarj3.cpp` ja kokeile toimintaa. Pitäisi toimia. Muuta ohjelma vielä seuraavaksi “kunnon” C++ –ohjelmaksi:

```
/* Ohjelma järjestää kolme lukua suuruusjärjestykseen.*/
#include <iostream.h>

/* Aliohjelma vaihtaa kaksi lukua keskenään. */
void vaihda(int &a, int &b)
{
  int t;
  t = a; a = b; b = t;
}

int main(void)
{
  int x1,x2,x3;

  cout << "Anna 1. luku >"; cin >> x1;
  cout << "Anna 2. luku >"; cin >> x2;
  cout << "Anna 3. luku >"; cin >> x3;

  if ( x1 > x2 ) vaihda(x1,x2);
  if ( x1 > x3 ) vaihda(x1,x3);
  if ( x2 > x3 ) vaihda(x2,x3);

  cout << "Luvut: " << x1 << " " << x2 << " " << x3 << endl;
  return 0;
}
```

4.7 Taulukot

Miten edellistä ohjelmaa pitäisi muuttaa mikäli haluttaisiin järjestää 4 lukua. Taitaisi jo tulla varsin sotkuiseksi.

1. Muutetaan ohjelmaa (kirjoita nimelle `JARJ3TA.C` käyttäen pohjana `JARJ3.C`-ohjelmaa) siten, että apuna käytetään taulukkoa:

```
/* Ohjelma järjestää kolme lukua suuruusjärjestykseen. */
#include <stdio.h>
#define KOKO 3
/* Aliohjelma vaihtaa kaksi lukua keskenään. */
void vaihda(int *a, int *b)
{
  int t;
  t = *a; *a = *b; *b = t;
}
int main(void)
{
  int luvut[KOKO];
  int i,pienin,tutki;
  for (i=0; i<KOKO; i++) {
    printf("Anna %d. luku >",i+1); scanf("%d",&luvut[i]);
  }
  for (pienin=0; pienin<KOKO-1; pienin++)     /* pienin aina paikkaan 0,1,2..*/
    for (tutki=pienin+1; tutki<KOKO; tutki++) /* etsitään osataulukon pienin */
      if ( luvut[pienin]>luvut[tutki] )
        vaihda(&luvut[pienin],&luvut[tutki]);
  printf("Luvut: ");
  for (i=0; i<KOKO; i++)
    printf(" %d",luvut[i]);
  printf("\n");
  return 0;
}
```

2. Aliohjelma `vaihda` voidaan hakea tiedostosta `JARJ3A.C` seuraavasti:
    a. Siirry tiedostoon `JARJ3A.C` (***`[Alt-ikkunan nro]`***).

   b. Merkitse aliohjelma lohkoksi (siirry alkuun ja ***[Shift]*** pohjaan, siirry nuolilla loppuun ja ***[Shift]*** ylös)
   c. Laita lohko leikekirjaan (***[Ctrl-Ins]***).
   d. Siirry tiedostoon `JARJ3TA.C`.
   e. Kopioi lohko leikekirjasta (siirry haluttuun kohtaan ja paina ***[Shift-Ins]***).
3. Edellinen ohjelma ei ehkä ole yhtä selkeä (?) kuin JARJ3.C, mutta lukujen lukumäärä esiintyy ohjelmassa enää vain yhdessä paikassa. Missä?
4. Talleta ohjelma.
5. Poista WATCH-ikkunasta kaikki tarpeeton:
   a. siirry WATCH-ikkunaan (***[Alt-W]W***)
   b. siirry poistettavalle riville
   c. poista rivi (***[Ctrl-Y]*** tai ***[Del]***)
   d. …
   e. siirry takaisin C-ikkunaan (***[Alt-#]***, missä # on sen ikkunan numero jossa haluttu C-ohjelma on.)
6. Lisää taulukko `luvut` sekä muuttujat `pienin` ja `tutki` WATCH-ikkunaan.
7. Aja ohjelmaa askel kerrallaan. Toimiiko täsmälleen vastaavasti kuin `JARJ3.C`?
8. Muuta `KOKO 5` ja kokeile toimintaa.

4.8 C-kielen kirjasto

Järjestämisessä voimme C-kielen tapauksessa käyttää apuna myös valmista kirjastorutiinia `qsort`. Rutiinin käyttämistä varten meidän tarvitsee vain kirjoittaa aliohjelma, joka osaa verrata järjestettävän taulukon kahta alkiota keskenään palauttaen negatiivisen luvun, nollan tai positiivisen luvun sen mukaan, miten verratattavat alkiot suhtautuvat toisiinsa:

1. Muuta edellinen ohjelma seuraavanlaiseksi ja talleta nimellä `jarjta.c`:

```
/* Ohjelma järjestää kolme lukua suuruusjärjestykseen. */
#include <stdio.h>
#include <stdlib.h>
#define KOKO 3

/* Alkioiden vertailufunktio.                          */
int sort_function(const void *a, const void *b)
{
  return *(int *)a - *(int *)b;
}
int main(void)
{
  int i,luvut[KOKO];
  for (i=0; i<KOKO; i++) {
    printf("Anna %d. luku >",i+1); scanf("%d",&luvut[i]);
  }
  qsort(luvut,KOKO,sizeof(luvut[0]),sort_function);
  printf("Luvut: ");
  for (i=0; i<KOKO; i++)
    printf(" %d",luvut[i]);
  printf("\n");
  return 0;
}
```

2. Aja ohjelmaa askel kerrallaan. Toimiiko?

3. Laita keskeytyskohta riville

```
  return *(int *)a - *(int *)b;
```

sekä lisää muuttujat `a` ja `b` WATCH-ikkunaan. Aja ohjelmaa `Run`-käskyllä. Monta kertaa aliohjelmassa käydään?

4. Kirjoita vielä edellisestä korjaillen seuraava ohjelma nimelle `jarjtar.c`:

```
/* Ohjelma järjestää kolme lukua suuruusjärjestykseen. */
#include <stdio.h>
#include <stdlib.h>
#define KOKO 3000

/* Alkioiden vertailufunktio.                          */
int sort_function(const void *a, const void *b)
{
  return *(int *)a - *(int *)b;
}

/* Aliohjelmalla tulostetaan taulukon alkuosa          */
void tulosta(int luvut[])
{
  int i;
  printf("Luvut: ");
  for (i=0; i<min(KOKO,10); i++)
    printf(" %5d",luvut[i]);
  printf("\n");
}
int main(void)
{
  int i,luvut[KOKO];
  for (i=0; i<KOKO; i++) {
    luvut[i] = rand();
  }
  tulosta(luvut);
  qsort((void *)luvut,KOKO,sizeof(luvut[0]),sort_function);
  tulosta(luvut);
  return 0;
}
```

5. Aja ohjelma ja ota aika kauanko ohjelman suoritukseen menee.
6. Kirjoita vastaava versio ohjelmasta `jarj3ta.c` ja kokeile kauanko sen suorittamiseen menee. Miksi?

4.9 Tavoite

1. Edellisistä ohjelmista `JARJ3A.C` näytti ehkä kaikkein selkeimmältä. Siinä oli kuitenkin se vika, että muuttaminen useamman luvun käsittelyä varten vaatisi suuria muutoksia itse ohjelmaan.
2. `JARJ3TA.C` voidaan helposti muuttaa käsittelemään useampaakin lukua. Se ei kuitenkaan ole sellaisenaan turhan selkeä.
3. Jatkossa pyritään siihen, että em. ohjelma voitaisiin kirjoittaa muodossa:

```
...
{ /* Pääohjelma */
  lue(luvut);
  jarjesta(luvut);
  tulosta(luvut);
  return 0;
} /* Pääohjelma */
```

4.10 Kerho-ohjelma 1. versiot

4.10.1 paamenu.c

1. Mene omaan hakemistoosi ja luo sen alle hakemisto nimeltä `KERHO` ja vaihda itsesi sinne.
2. Kopioi KERHO-ohjelman 0. versio omaan hakemistoosi:

```
C:\OMAT\OHJELMOI\VESA>copy n:\kurssit\cpp\kerho\paamenu.0\*.*[RET]
```

3. Tutki Borland-C:llä tiedostoa `kerho.c`
4. Laita tulosikkunan näkyville.
5. Aja ohjelmaa kokonaisena ja askel kerrallaan. Missä kohti tulostuu jäsenien lukumäärä?
6. Muuta ohjelmaa siten, että se tulostaa jäsenien määräksi 15.
7. Ymmärrätkö kaikki rivit?

5. Pääteohjaus 3

5.1 #define

Tutki (=kirjoita lyhyt malliohjelma, jolla voit testata) pitävätkö seuraavat väitteet paikkansa (kirjoita vastaukset väitteiden viereen ja tarkista lopuksi ohjaajan kanssa väitteet):

1. Myös lainausmerkeissä oleva teksti korvautuu!
2. #define pitää kirjoittaa ennen kaikkia aliohjelmia!
3. #define on muotoa #define vakio=arvo
4. #define rivi pitää päättää puolipisteeseen.
5. Seuraavasta seuraa käännösvirhe:

```
#define OMA 5;
...
  a = a + OMA + 3;
```

6. #define korvaa vain kokonaisia sanoja.
7. #define ei voi sisältää funktiokutsuja.
8. #definellä ei voi määritellä tyhjiä jonoja

5.2 Funktiot

Kirjoita seuraavat funktiot tai aliohjelmat sekä testiohjelma, jolla testataan niiden toimivuus.

1. Funktio `int etumerkki(double x, double eps)` joka palauttaa

```
-1  mikäli x<-eps
 0  mikäli -eps<=x<=eps
 1  mikäli x>eps
```

2. Aliohjelma `kysy_vuosi`, joka kysyy vuosiluvun ja palauttaa sen parametrinaan. Aliohjelman nimessä palautetaan 0 mikäli annettu vuosiluku ei ole tältä vuosisadalta tai syöttö on muuten väärin. Mikäli syöttö on tältä vuosisadalta, palautetaan aliohjelman nimessä 1.

5.3 Kerho-ohjelma 1. versiot

5.3.1 Runko

1. Kopioi KERHO-ohjelman 1. versio omaan hakemistoosi:

```
C:\OMAT\OHJELMOI\VESA>copy n:\kurssit\cpp\kerho\runko.1\*.*[RET]
```

2. Tutki Borland-C:llä tiedostoa `kerho.c`
3. Aja ohjelmaa kokonaisena ja askel kerrallaan.
4. Ymmärrätkö nyt ***[F7]*** ja ***[F8]*** eron?
5. Siirry riville, jossa lukee

   ```
   nimi = "Ankka Aku";
   ```

6. Laita riville keskeytyskohta (***[Ctrl-F8]***).
7. Aja ohjelma (***[Ctrl-F9]***).
8. Vastaile kunnes ohjelma pysähtyy.
9. Laita watch ikkunaan muuttujat `this` ja `*this`.
10. Aja askel kerrallaan ja seuraa mitä tapahtuu.
11. Tutki mitkä osat ymmärrät (vrt. listaus monisteessa).

5.4 Taulukot

1. Kirjoita funktio joka palauttaa nimessään C-merkkijonon pituuden. Kirjoita testiohjelmaksi vakiokutsuja:

   ```
   pit=pituus("Aku Ankka");
   ...
   ```

2. Kirjoita funktio, joka palauttaa nimessään parametrina annetun C-merkkijonon välilyöntien lukumäärän. Kirjoita testiohjelmaksi vakiokutsuja:

   ```
   val=valeja("Aku Ankalla on merimiespusero päällä.");
   ...
   ```

6. Pääteohjaus 4

Tässä ohjauksessa harjoitellaan osiinjaetun tiedoston kääntämistä ja itse valmiiksi kirjoitettujen aliohjelmakirjastojen käyttöä.

6.1 Projekti

1. Mene omaan hakemistoosi KOVALEVYLLÄ ja luo sen alle hakemisto nimeltä `KERHO` ja vaihda itsesi sinne.
2. Kopioi KERHO-ohjelma omaan hakemistoosi:

   ```
   C:\OMAT\OHJELMOI\VESA\KERHO>copy n:\kurssit\cpp\kerho\etsilaj.5\*.*[RET]
   ```

3. Käynnistä Borland-C (***BC[RET]***).
4. Aloita uusi projekti (***[Alt-P]*** ja Open Project).
5. Anna projektin nimeksi ***KERHO***.
6. Avaa tiedosto `kerhomai.cpp` ja laita projektiin ne tiedostot, jotka tiedoston `kerhomai.cpp` kommenteissa pyydetäään laittamaan projektiin (lisäämistä varten paina ***[Ins]***):

   ```
   tulkitse esim CPP\ALI\MJONOT.C =>  N:\KURSSIT\CPP\ALI\MJONOT.C
   ```

7. Katso, että kohdan Options|Directories kohdassa Include Directories lukee (älä hävitä rivin alkua!):

   ```
   ....INCLUDE;N:\KURSSIT\CPP\ALI
   ```

   Jollei lue, niin lisää viimeinen hakemisto.
8. Kokeile ajaa ohjelmaa (valmiina on kerho KELMIT). Toimiiko kuten suunniteltu?

6.2 Breakpoint

Otetaan lyhyt debuggerin kertausharjoitus. Debuggerilla ohjelmaa pystyttiin ajamaan mm. askel kerrallaan. Oikeassa ohjelmassa kukaan ei jaksa kahlata koko ohjelmaa tällä tavoin läpi. Pitää keskittyä vain olennaisuuksiin.

Tämän vuoksi debuggerissa on mahdollisuus laittaa keskeytyskohtia, eli paikkoja joihin tultaessa pysähdytään, muuten ohjelmaa ajetaan normaalisti. Keskeytyskohtia voidaan laittaa myös ehdollisesti hyvinkin monipuolisesti. Kaikkein hienoimmissa systeemeissä keskeytys saadaan jonkin tietyn muuttujan saadessa tietyn arvon (tai yleensä muuttuessa) missä tahansa kohti ohjelmaa.

1. Epäilet, että metodi `cNaytto::kysy_tiedot` ei toimi. Toimimattomuus ilmenee (ilmenisi esimerkiksi, koska nyt toimii) jotenkin henkilön lisäämisessä. Epäilet erityisesti aliohjelmaa kysy_kentta (`NAYTTO.CPP`).
2. Avaa tiedosto `NAYTTO.CPP` (esimerkiksi näyttämällä tiedoston nimeä projekti-ikkunassa tai painamalla ***[F3]***).
3. Etsi aliohjelma `kysy_kentta` (***[Ctrl-Q]F kysy_kentta[RET]***). Mikäli ei löydy 1. yrittämällä, niin ***[Ctrl-L]*** etsii uudelleen.
4. Siirry aliohjelmassa riville jolla lukee

   ```
   if ( paluu < OLETUS ) return TOIM_LOPETA;
   ```

   ja laita riville keskeytyskohta (breakpoint, joko ***[Ctrl-F8]*** tai ***[Alt-D]*** ja Toggle breakpoints).
5. Lisää watch-ikkunaan aliohjelman kannalta mielenkiintoisia arvoja:

   ```
   paluu
   toim
   apu
   jono
   viesti
   ```

6. Aja ohjelma ***[Ctrl-F9]*** siten, että yrität lisätä jotakin tietoa. Kun ohjelma pysähtyy, aja askel kerrallaan joko suorittaen tai ylittäen aliohjelmat. (Huom! Koska `mjonot.c` tiedostoa ei kannata debuta, pitää sinne kuuluvat kutsut ajaa ***[F8]***-nappulalla!). Seuraa muuttujien arvon käyttäytymistä. Kun olet “steppaillut” ulos epäilemästäsi aliohjelmasta paina taas ***[Ctrl-F9]***. Jatka tätä kunnes “Olet saanut vian selville”. Sitten voit poistaa keskeytyskohdan vastaavasti kuin sen asetitkin. Huom! Keskeytyskohtia voi olla useitakin.
7. Kokeile joitakin muita kohtia vastaavasti (mielellään sellaisia, joista itse olet kiinnostunut, esimerkiksi metodia `lue_tiedostosta` tiedostossa `kerho.cpp`).
8. Näin voit aina menetellä, mikäli ohjelmastasi löytyy toimimattomia (tai epäilyttäviä) kohtia jonkin pitkän rämeikön takaa. Muista sitten joskus lopuksi myös poistella keskeytyskohdat!

6.3 lue_jono_oletus

Kokeillaan `lue_jono_oletus` aliohjelman toimintaa.

1. Sulje kaikki ikkunat.
2. Aloita uusi projekti nimellä `lue`.
3. Lisää projekti-ikkunaan vielä kirjoittamaton tiedosto `lue.c`.
4. Lisää myös `n:\kurssit\cpp\ali\mjonot.c` (voit myös kopioida aliohjelmat omaan hakemistoosi ja laittaa vain `mjonot.c`).
5. Kirjoita seuraava pääohjelma tiedostoon `lue.c`:

   ```
   #include <stdio.h>
   #include "mjonot.h"
   int main(void)
   {
     char nimi[40]="Minä";
     int paluu;
     paluu = lue_jono_oletus("Anna nimesi",17,33,nimi,nimi,sizeof(nimi));
     if (paluu == OLETUS) printf("Et muuttanut nimeä!\n");
     printf("Nimesi oli %s\n",nimi);
   ```

```
  return 0;
}
```

6. Jos ohjelma ei käänny, niin tarkista että kohdan Options|Directories kohdassa Include Directories lukee:

   ```
   ....INCLUDE;N:\KURSSIT\CPP\ALI
   ```

   Jollei lue, niin lisää viimeinen hakemisto.
7. Kokeile käyttää ohjelmaa siten, että kirjoitat joskus uuden nimen ja joskus vastaat pelkän ***[RET]***.
8. Muuta ohjelmaa siten, että `nimi` muutetaan isoiksi kirjaimiksi ennen tulostamista ja kokeile.
9. Muuta ohjelmaa siten, että `nimi` muutetaan alku-isoiksi kirjaimiksi ennen tulostamista ja kokeile.
10. Muuta ohjelmaa siten, että kaikki turhat välilyönnit poistetaan ja kokeile.
11. Kokeile mitä merkitystä on parametrien 17 ja 33 muuttamisella.
12. Kokeile korvata ensimmäinen kutsussa oleva `nimi`-parametri vakiolla `"Kalle"`.
13. Poistu käyttöjärjestelmään ja kopioi hakemiston `n:\kurssit\cpp\ali` sisältö omaan hakemistoosi.
14. Käynnistä jälleen Borland-C ja luo nyt uusi projekti nimelle `lue2`.
15. Lisää projektiin tiedostot `lue.c` ja `mjonot.c` (ei tarvitse nyt polkua).
16. Käännä ohjelma. Siis aliohjelmakirjastot voi joko kopioida omaan hakemistoon tai käyttää tietystä hakemistosta. Levyaseman nimeä ei kuitenkaan mielellään saa laittaa polkuihin!

6.4 string-luokka

1. Tee uusi projekti `luecpp`, lisää siihen `lue.cpp`:

   ```
   #include <iostream.h>
   #include "mjonotpp.h"
   int main(void)
   {
     string nimi("Minä");
     int paluu;
     paluu = lue_jono_oletus("Anna nimesi",17,33,nimi);
     if (paluu == OLETUS) cout << "Et muuttanut nimeä!\n";
     cout << "Nimesi oli " << nimi << endl;
     return 0;
   }
   ```

2. Lisää vielä projektiin `mjonot.c` ja `mjonotpp.cpp` (sekä `string.cpp` jos käytät jotakin vanhempaa kääntäjää < BC 4.5?)
3. Muuten tee samat tehtävät kuin edellisessä kohdassa.

6.5 pvm

1. Mene käyttöjärjestelmään ja kopioi hakemiston `n:\kurssit\cpp\demot` sisältö omaan hakemistoosi.

2. Käynnistä Borland-C ja luo projekti nimeltä `pvm`. Lisää projektiin tiedostot `t_pvm.c` ja `pvm.c`.
3. Katso, että tiedosto `t_pvm.c` on seuraavanlainen seuraavasti (korjaa jollei ole):

```
#include <stdio.h>
#include "pvm.h"
...
int main(void)
{
/*  tayta_kk_1(KK_1); */
  testaa_kv();
/*  testaa_pvm(); */
  return 0;
}
```

4. Kokeile ajaa ohjelmaa.
5. Korjaa tarvittaessa tiedostoa `pvm.c`.
6. Kokeile tehdä demotehtäviä ja tee niitäkin varten jonkinlainen testipääohjelma.

6.6 avustus (help)

1. Lue avustuksen käyttämisen ohjeet tiedostosta `help.c`.
2. Tee pieni (alle 10 riviä) testipääohjelma avustuksen testaamiseksi.
3. Kirjoita pieni (muutama aihe) avustustiedosto.
4. Kokeile ohjelman toimintaa.

6.7 Leikkaaminen (cut tai copy) ja liimaaminen (paste)

Aikaisemmissa pääteohjauksessa opeteltiin merkitsemään lohkoja ja siirtämään tai kopioimaan niitä paikasta toiseen.

Helpointahan tämä oli, mikäli käytössä on editori, jolla voidaan pitää useita ikkunoita auki yhtäaikaa.

Tällöin toimittiin seuraavasti:

1. Siirry ikkunaan josta haluat leikata.
2. Maalaa alue:
   a. joko alueen alkuun ja ***[Ctrl-K]B*** ja alueen loppuun ***[Ctrl-K]K***
   b. tai alueen alkuun tai loppuun, paina ja pidä ***[Shift]*** pohjassa ja siirry alueen toiseen kulmaan
   c. tai maalaa hiirellä siirtymällä alkuun ja painamalla ja pitämällä näppäin pohjassa siirrytään alueen toiseen kulmaan
3. laita lohko leikekirjaan
   [Ctrl-Ins] = kopioi
   [Shift-Del] = leikkaa
4. Siirry ikkunaan (tai paikkaan) jonne haluat tuloksen.
5. “Liimaa” leikekirjan sisältö haluttuun paikkaan ***[Shift-Ins]***

Mikäli leikekirjaa tai useita ikkunoita ei ole, voidaan aina menetellä seuraavasti:

1. Siirry tiedostoon josta haluat palan.
2. Merkitse lohko.
3. Kirjoita lohko levylle vaikkapa nimelle `A.` (***`[Ctrl-K]WA.[RET]`***)
4. Siirry tiedostoon jonne haluat tuloksen.
5. Lue tiedosto `A.` levyltä (***`[Ctrl-K]RA.[RET]`***).

Samassa tiedostossahan lohko pystyttiin siirtämään tai kopioimaan:

1. Merkitse lohko
2. Laita lohko leikekirjaan (kopioi tai leikkaa)
3. Siirry kohtaan johon haluat lohkon.
4. Paina ***`[Shift-Ins]`*** (liimaa).

Tietysti siirto tai kopiointi käy hienosti myös leikekirjan kautta, mikäli leikekirja on olemassa. Leikekirjan käytössä on vielä se etu, että leikekirjan sisältö säilyy muuttumattomana kunnes sinne laitetaan jotain uudestaan.

6.7.1 Merkitse rivin alusta seuraavan rivin alkuun

Yleensä lohkot kannattaa merkitä siten, että alkupaikka merkitään RIVIN ALKUUN (***`[Home]`***) ja loppupaikka sen rivin alkuun, jota ei enää lohkoon haluta. Näin merkittynä lohko sisältää mukavasti kaikki tarvittavat rivinsiirrotkin!

Tietysti jos haluat leikata vaan osan riviä (esimerkiksi inhottavan muuttujan nimen), niin silloin merkitään tasan haluttu osa!

Sisällys