

## Modulaarinen ohjelmointi

Luento 9  
7.10.2008

## Sisältö

- n Modulaarisuus C:ssä
- n Käännösyksikkö ja otsaketiedosto
- n Makefile

## Ison ohjelman toteuttaminen

- n Kokonaisuus on jaettava hallittaviin osiin
  - n Toiminnallisia kokonaisuuksia (Java: luokat)
    - n Syöttö ja tulostus
    - n Virheiden käsittely
    - n Tietyn toiminnan tai rajatun tehtävän toteuttaminen
  - n Osien väliset rajapinnat eli parametrit ja paluuarvot
    - n Selkeästi määriteltyjä
    - n Vähän riippuvuuksia eri osien välillä

Modulaarisuus on keino hallita monimutkaisuutta!  
Moduulit toteuttavat abstraktion, kapseloinnin ja  
informaation piilottamisen.

## Modulaarisuuden hyödyt

- n Jako osiin + selkeät ja yksinkertaiset rajapinnat => koko ohjelman rakenne selkeämmäksi
- n Ohjelmisto voidaan toteuttaa projektityönä: eri henkilöt ohjelmoivat eri osat
- n Testaus voidaan aloittaa hyvin varhaisessa vaiheessa
- n Ylläpito helpottuu: muutokset vain tarvittaviin kohtiin eli joihinkin funktioihin
- n Koodin uudelleenkäyttö: standardikirjastot, omat kirjastot
  - n Kerran ratkaistu ja koodattu, voidaan käyttää monta kertaa

## Modulaarinen ohjelmointi ja C-kieli

- n C ei varsinaisesti tue modulaarista ohjelmointia (vertaa esim. Modula)
- n C:ssä on piirteitä, joiden avulla ohjelman modulaarisuus voidaan toteuttaa
  - n funktiot ja niiden protyytit
  - n otsaketiedostot (header files)
- n Näitä piirteitä sopivasti käyttämällä saadaan C:ssä toteutettua modulaarinen ohjelma tehokkaasti

n Toiminnon toteutus ○ toiminnon käyttö

Piilotetaan käyttäjältä

tarkat ohjeet käyttämiseen

## C:n piirteitä modulaariseen ohjelmointiin

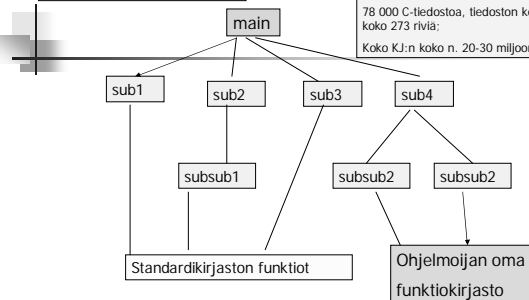
- n funktiot
  - n Ohjelma jaetaan useaksi pienehköksi funktioksi, joista kukin suorittaa tietyn toimenpiteen
    - n – noin 20 riviä kommentteineen on sopiva funktion koko, sillä näkyy helposti kokonaan näytöllä
    - n Valmiit kirjastofunktiot: standardikirjaston funktiot tai omat kirjastofunktiot käytettävissä
  - n Pääohjelma main lähinnä koostuu funktiokutsuista
  - n Lisähierarkia: funktiot voivat kutsua toisia funktioita
- n funktioiden esittelyt eli protyytit
  - n Funktio on määriteltävä tai esiteltävä ennen käyttöä
  - n Kääntäjä voi tarkistaa oikean käytön
- n otsaketiedostot
  - n Sisältävät tietoja, joita ohjelmat tarvitsevat voidakseen käyttää muualla määriteltyjä funktioita
    - n #include <stdio.h>

## Pienehkön ohjelman modulaarinen rakenne

- n #include-määrittelyt kirjastofunktioiden liittämiseksi
  - n voivat sisältää toisia includeja
- n vakiomäärittelyt, tyyppiesittelyt
  - n helpompi hallita: löytää ja muuttaa
- n Ohjelman funktioiden prototyypit eli niiden esittelyt
  - n Näin määritellään ennen käyttöä
- n Pääohjelma main (joka ohjelmassa ainakin yksi)
  - n ja sen funktiokutsut
- n Ohjelman funktioiden määrittelyt

Entä kun funktioita on hyvin paljon ja niitä toteuttavat eri projektiyhmät ja useat ihmiset?

Linux: ytimessä 2.4 miljoonaa koodiriviä; tasta 1.4 milj. riviä eri laitteiden ajureita varten  
78 000 C-tiedostoa, tiedoston keskim. koko 273 riviä;  
Koko KJ:n koko n. 20-30 miljoona riviä



Modulaarisuus funktioiden tasolla

## Isohko ohjelma tai projekti

- n Yhteenkuuluvat funktiot omiksi tiedostoiksi eli käännösyksiköiksi => moduuli
  - n Linuxin koodi
- n Kukin käännösyksikkö voidaan kääntää ja testata erikseen
  - n => objektimoduuli
- n Ajettava ohjelma saadaan, kun erikseen käännetty objektimoduulit linkitetään yhteen
- n Erikseen kunkin moduulin lähdekooditiedosto ja otsaketiedosto => joustavuutta kääntämisessä
  - n Ei tarvitse kääntää turhaan objektimoduulin koodia
  - n Otsaketietojen liittäminen riittää

## Kooditiedostot muodostavat C:n moduulin

- n C:n funktiot ovat globaaleja, kaikkialta ohjelmasta kutsuttavissa
  - n Eivät sellaisenaan yhdessä tiedostossa tarjoa
    - n riittävää kapselointia
    - n toteuttamisen, muuttamisen, kääntämisen ylläpidon joustavuutta
- n Toteutuksen ja käytön erottava moduuli syntyy, kun
  - n kootaan joukko yhteenliittyviä funktioita samaan kooditiedostoon (jokunimi .c) = moduulin toteutus
  - n ja näiden funktioiden prototyypit ja muu muiden moduulien käyttöön tarkoitettu data -otsaketiedostoon (jokunimi.h) = moduulin rajapinta

## Kapselointi: static -

- n Määrittelee talletusluokkaa ja näkyvyyttä (scope)
  - n Paikalliset muuttujat
    - n Funktion paikallisille muuttujille varataan tilat pinosta. Ne ovat käytettävissä vain funktion suoritusajan.
    - n static-määrittely => vain funktion sisällä käytettävissä, mutta säilyttävät arvonsa käyttökerrasta toiseen (esim.käyttökertalaskuri)
  - n Globaalit muuttujat ja funktiot
    - n static-määrittely rajaa näkyvyyden siihen tiedostoon, jossa ne ovat määriteltä
    - n Mahdollistaa datan piilottamisen

## Kapselointi:

"estä tiedon vuotaminen ulkopuolelle"

~ Javan private

- n Kapseloidaan moduulin muuttujia ja funktioita
  - n static => eivät ole käytettävissä moduulin ulkopuolella
    - n Hyvä käytäntö jotenkin erottaa nimestä tällaiset muuttujat ja funktiot
  - n funktiota edeltää static-määre
    - n static void e\_append\_0; /\* kuuluu editor-moduuliin \*/
  - n Globaalia muuttujaa edeltää static-määre
    - n static int e\_flag\_ = 0; /\* tämä samoin\*/
  - n static muuttaa sen, kuinka linkittäjä käsittelee muuttujaa
- n Huom! C:ssä kapselointiyksikkönä on tiedosto! Javassa objekti.

## extern-määrittely

- n Funktioiden esittelyssä oletusarvona => voidaan jättää pois
- n Yhteiskäyttöiset muuttujat
  - n Harvoin todella tarpeen ja hyödyllisiä
- n extern int myErrorNo
  - n Linkittäjä: kaikki viittaukset kohdistuvat samaan muuttujaan, vaikka olisivat eri tiedostoissa
  - n Globaalin muuttujan static-määrittely => vain samassa tiedostossa kaikki viittaukset kohdistuvat samaan muuttujaan. Toisen tiedoston viittaukset kohdistuvat toiseen muuttujaan.

## Vaikutus linkittäjän toimintaan

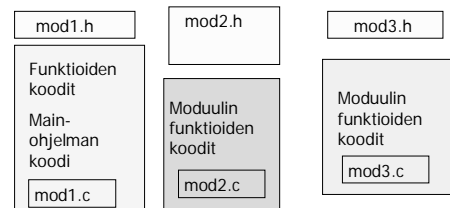
- n Ohjelmassa voi olla useita samannimisiä muuttujia (esim. i, j) ja linkittäjän täytyy tietää, milloin kyseessä on viite yhteen ja samaan muuttujaan ja milloin taas eri muuttujiin
- n muuttujien linkkiytymistapa (linkage)
  - n ulkoinen (external): ulommalla tasolla määritelty
    - n Yhteiskäyttöisiä, usean tiedoston sisällä viittaavat aina yhteen ja samaan olioön
  - n ei mitään: funktion sisällä määritelty
    - n Linkittäjän kannalta aina eri olioita
  - n sisäinen (internal): const-määritelty muuttujat, rakenteiset tyypit (struct, union, enum)
    - n Yhden tiedoston sisällä viittavat aina samaan olioön, eri tiedostoissa eri olioihin
- n static ja extern muuttavat linkkiytymistapaa
  - n static muuttaa ulkoiseksi
  - n extern muuttaa ulkoiseksi

## Globaalit muuttujat

Javassa ei ole  
globaaleja muuttujia!

- n Ohjelmassa funktion ulkopuolella määritelty muuttujat ovat globaaleja
  - n koko ohjelmassa käytettävissä määrittelynsä jälkeen
  - n elinikä sama kuin koko ohjelmalla
  - n alustetaan nolaksi
- n C:ssä tulee käyttää hyvin harkiten ja pyrkiä välttämään käyttämistä
  - n Funktiokutsut eivät saa muuttaa globaaleja muuttujia
    - n Seuraa vaikeuksia: testaus, virheiden jäljitys, ylläpito
    - n Dokumentoitava huolellisesti
- n Määriteltävä yhdessä käännösyksikössä ja liitettävä .h-tiedostoon (extern int globalvar)

## Käännösmoduulit



## Kooditiedosto (.c)

- n Sisältää yhteenkuuluvien, jollakin tavoin samaan kokonaisuuteen liittyvien funktioiden koodin
  - n Kukin funktio suorittaa jonkun tietyn toiminnon esim.
    - n syöttötietojen käsittelyn
    - n tulostukset
    - n varsinaisen tietojen käsittelyn ja muokkaamisen:
      - n laskenta, lajittelu, jne
    - n virheiden käsittelyn
    - n jne
- n Yhdessä kooditiedostossa main
- n Muiden moduulien käyttöön tarkoitettu informaatio kootaan otsaketiedostoksi
  - n Pyritään pitämään suhteellisen pienenä
- n Eri moduulit suhteellisen riippumattomiksi toisistaan

Vastaa Javan rajapintaa (interface)

## Otsaketiedosto

- n Sisältää:
  - n funktioiden esittelyt (prototyypit)
  - n Makrot
  - n vakioiden määrittelyt (const)
  - n Dokumentoinnin: kaikki mitä käyttäjän tarvitsee tietää osatakseen käyttää
    - n Ei mitään pelkästään toteutukseen liittyvää
- n esikäantäjä liittää otsaketiedoston moduuliin
  - n #include <stdio.h>
    - n standardikirjaston funktion prototyypit moduuliin
  - n #include "oma.h"
    - n Samassa hakemistossa olevan oman otsaketiedoston liittäminen

Header files usually ONLY contain definitions of data types, function prototypes and C preprocessor commands.

## makefile

- Ohjelmassa on useita moduuleja. Kukin moduuli eli käännösyksikkö omassa tiedostossaan, kirjastofunktiot omassa tiedostossaan
- Käännösyksiköt käännetään erikseen ja linkitetään sitten yhteen
  - `gcc -c main.c` tekee objektitiedoston `main.o`
  - `gcc -c mod1.c` => `mod1.o`
  - `gcc -c mod2.c` => `mod2.o`
- `gcc -o ohjelma main.o mod1.o mod2.o` linkittää objektitiedostot suoritettavaksi ohjelmaksi

```

/* main.c */
#include <stdio.h>
#include "mod1.h"
#include "mod2.h"
int main(void) {
    mod1();
    mod2();
    return 1;
}

/* mod1.c */
#include <stdio.h>
#include "mod1.h"
void mod1(void){
    .....
    puts("moduuli yksi");
    .....
}

/* mod2.c */
#include <stdio.h>
#include "mod2.h"
void mod2(void){
    .....
    puts("moduuli kaksi");
    .....
}

/* mod1.h */
void mod1(void);

/* mod2.h */
void mod2(void);

gcc -c main.c
gcc -c mod1.c
gcc -c mod2.c
gcc -o ohjelma main.o mod1.o mod2.o
    
```

## Moduulien kääntäminen – make

- Kääntämiskomennot ja ohjeet kirjoitetaan säännöiksi makefile tai Makefile nimiseen tiedostoon
  - kohde: tarvittavat tiedostot
  - komento1
  - komento2
  - ....
  - komentoz
- Huom. Komentojen sisennykseen käytetään tabulaattorimerkkiä (tabulointia), ei välilyöntejä.

## makefile:n laatiminen

```

# kommenttirivi

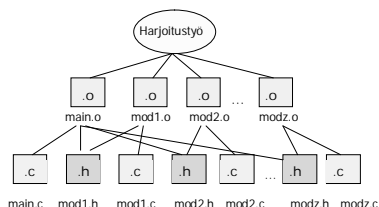
gcc -c main.c
gcc -c mod1.c
gcc -c mod2.c
gcc -o ohjelma main.o mod1.o mod2.o

# makefile
CC = gcc -ansi -pedantic -Wall
ohjelma: main.o mod1.o mod2.o
$(CC) -o ohjelma main.o mod1.o mod2.o
mod1.o: mod1.c mod1.h
$(CC) -c mod1.c
mod2.o: mod2.c mod2.h
$(CC) -c mod2.c
main.o: main.c mod1.h mod2.h
$(CC) -c main.c

http://www.cs.helsinki.fi/group/sqtr/makefile_ohje.html
http://www.eng.hawaii.edu/Tutor/Make/index.html
http://veritgo.hsr1.rutgers.edu/ug/make_help.html
    
```

Eräs skriptikieli, kuten  
mm  
-Job Control Languages  
-Unix Shell  
-QuakeC, AWK, perl, ...

## riippuvuusgraafit



- " Modular programming is a life saver, you can save an immense amount of time, if you do it right"