

## Modulaarinen ohjelmointi

### Kertaus osoittimista

Luento 9  
28.3.2006



## Sisältö

---

- n Modulaarisuus C:ssä
- n Käännösyksikkö ja otsaketiedosto
- n Makefile
- n Kertaus osoittimista ja funktioista
- n Kokeesta

## 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 määriteltäjä
    - n Vähän riippuvuuksia eri osien välillä

Modularisuus 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     $\leftrightarrow$     toiminnon käyttö

## C:n piirteitä modulaariseen ohjelmointiin

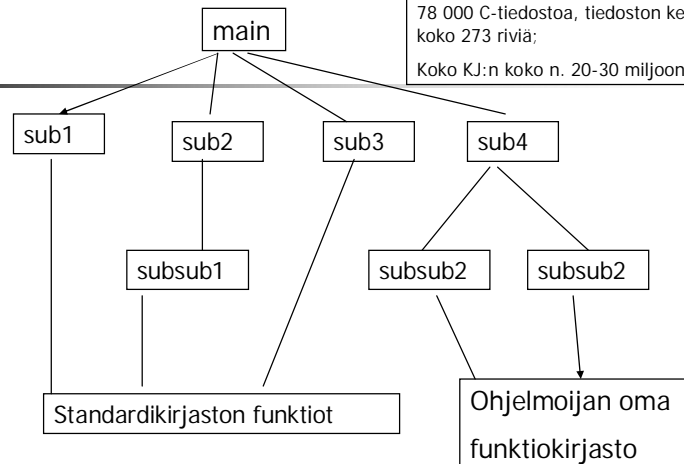
- n funktiot
  - n Ohjelma jaetaan useaksi pienehköksi funktioksi, joista kukin suorittaa tietyn toimenpiteen
    - n ~ noin 20 riviä kommentteineen on sopiva funktion koko, sillä näkyy helposti kokonaan näytöllä
    - n Valmiit kirjastofunktiot: standardikirjaston funktiot tai omat kirjastofunktiot käytettävissä
  - n Pääohjelma main lähinnä koostuu funktiokutsuista
  - n Lisähierarkiaa: funktiot voivat kutsua toisia funktioita
- n funktioiden esittelyt eli prototyypit
  - 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 prototyytit eli niiden esittely
  - 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 projektiryhmät ja useat ihmiset?

Linux: ytimessä 2.4 miljoonaa koodiriviä; tästä 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 käännetyt 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 ja ylläpidon joustavuutta
- n Toteutuksen ja käytön erottava moduuli syntyy, kun
  - n kootaan joukko yhteenliittyviä funktioita samaan kooditiedostoon (jokinimi .c) = moduulin toteutus
  - n ja näiden funktioiden prototyypit ja muu muiden moduulien käyttöön tarkoitettu data otsaketiedostoon (jokinimi.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 määritelty
    - 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(); /* kuuluu editor.h-tiedostoon */`
  - 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

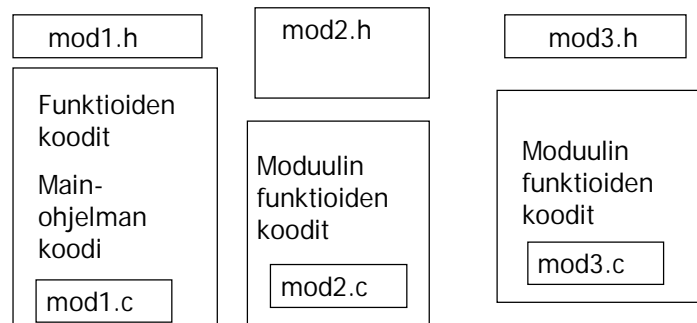
- n Ohjelmassa voi olla useita samannimisiä muuttujia (esim. i,j) ja linkittäjän täytyy tietää, milloin kyseessä on viite yhteen ja samaan muuttujaan ja milloin taas eri muuttujiin
- n muuttujien linkkiytymistapa (linkage)
  - n ulkoinen (external): ulommalla tasolla määritellyt
    - n Yhteiskäyttöisiä, usean tiedoston sisällä viittaavat aina yhteen ja samaan olioön
  - n ei mitään: funktion sisällä määritellyt
    - n Linkittäjän kannalta aina eri olioita
  - n sisäinen (internal): const-määritellyt 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 ulkoisen sisäiseksi
  - n extern muuttaa ulkoiseksi

Javassa ei ole  
globaaleja muuttujia!

## Globaalit muuttujat

- n Ohjelmassa funktion ulkopuolella määritellyt 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 käännoyksikössä 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 esikääntäjä 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

- n Ohjelmassa on useita moduuleja. Kukaan moduuli eli käännösyksikkö omassa tiedostossaan, kirjastofunktiot omassa tiedostossaan
- n Käännösyksiköt käännetään erikseen ja linkitetään sitten yhteen
  - n gcc -c main.c tekee objektitiedoston main.o
  - n gcc -c mod1.c => mod1.o
  - n gcc -c mod2.c => mod2.o
  - n 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

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

C-ohjelmointi  
Kevät 2006

Liisa Marttinen

21

## makefile:n laatiminen

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

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

Makefile kirjoitetaan vain kerran, mutta sitä voidaan käyttää monta kertaa komennolla **make**

- käännetään vain ne tiedostot, joita on muutettu
- linkitetään uudelleen vain, jos jokin linkitettävistä objektitiedostoista on muuttunut

C-ohjelmointi  
Kevät 2006

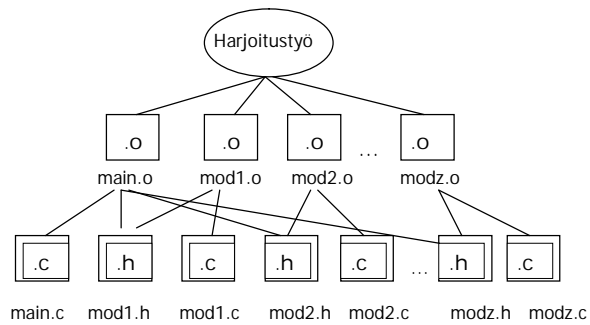
```
# kommenttirivi
#makefile
CC = gcc -ansi -pedantic -Wall
ohjelma: main.o mod1.o mod2.o
    $(CC) -o ohjelma main.o mod1.o mod2.o
mod1.o: mod1.c mod1.h
    $(CC) -c mod1.c
mod2.o: mod2.c mod2.h
    $(CC) -c mod2.c
main.o: main.c mod1.h mod2.h
    $(CC) -c main.c
```

[http://www.cs.helsinki.fi/group/sqltr/makefile\\_ohje.html](http://www.cs.helsinki.fi/group/sqltr/makefile_ohje.html)  
<http://www.eng.hawaii.edu/Tutor/Make/index.html>  
[http://vertigo.hsr1.rutgers.edu/ug/make\\_help.html](http://vertigo.hsr1.rutgers.edu/ug/make_help.html)

Liisa Marttinen

22

# riippuvuusgraafit



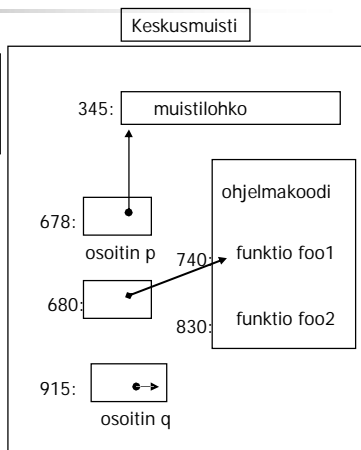
n " Modular programming is a life saver,  
you can save an immense amount of  
time, if you do it right"

# Lyhyt kertaus osoittimista

```
char *p; /* char, int, jne ilmoittavat, minkä tyyppisiä */
int *q; /* olioita sisältäviin muistilohkoihin osoittavat */
Nämä määrittelyt varaavat tilan vain osoittimelle!
```

```
char *p = "Tätä varten varataan muistitilaa";
int luvut[] = {1, 2, 3, 4, 5};
double taulu[100];
Nämä varaavat tilaa myös muistilohkelle sekä
asettavat osoittimen osoittamaan ko. muistilohkoa.
```

```
Tilaa voidaan varata myös malloc- ja calloc-
funktioilla ja samalla asettaa jokin osoitin
osoittamaan varattua muistilohkoa.
```



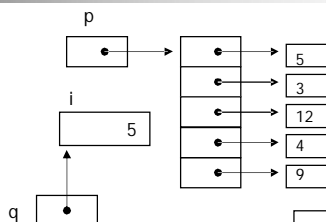
# Osoittimien käyttöä

```
n int **p;
int *q,r;
int i;

i = **p
```

```
q = &i; /* i:n osoite q:n arvoksi
i = *q+1; i = ++*q; /* i=6*/
i = *q++; /* ??? */
r = q; *r = 3; /* i=3 */
```

```
void *p; i = *(int*) p;
```



```
const int *p;
int const *p;
const int const *p;
```

```
char viesti [] = "On aika"; viesti: On aika\0
char *pv = "On aika"; pv: On aika\0
```

## Osoitin parametrina

x:n osoite, y:n osoite

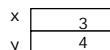
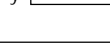
- n C:ssä vain arvoparametreja => funktio käyttää vain kopioita eikä voi mitenkään muuttaa saamiensa parametrien arvoja:

```
void swap(int x, int y) {
    int apu;
    apu=x;
    x=y;
    y=apu;
}
```

 kopioita

```
void swap(int *x, int *y) {
    int apu;
    apu=*x;
    *x=*y;
    *y=apu;
}
```

Kutsu: swap (&x, &y);

x  3  
y  4

double product (const double block, int size);

Varmistaa, ettei funktio muuta viiteparametrina saamaansa lohkoa

C-ohjelmointi  
Kevät 2006

Liisa Marttinen

27

## C: funktio-osoittimet

Java: "overriding"

- n Esimerkki: funktio voi suorituksen aikana vaihtaa käyttämänsä lajittelualgoritmia alkioden lukumäärän perusteella

`int (*fp) (void);`

`int *fp()`

Funktio, joka palauttaa int-tyyppisen osoittimen!

Osoitin int-tyypin palauttavaan funktioon

```
int fname(); /* kunhan funktio on oikean tyyppinen */
fp = fname; /* fp() merkitsee nyt samaa kuin fname()
```

- n Funktion parametrina funktio

```
void qsort(*line[], int left, int right, int (*comp)(void *, void*));
```

C-ohjelmointi  
Kevät 2006

Liisa Marttinen

28

```

void main (void) {
int choice; double x, fx;
funcptr fp;
.....
funcprt function[7] = {NULL, sin, cos, tan, log , log_2, exp}; /*määriteltyjä funktioita*/
/* funktiomenun tulostus: käyttäjää valitsee  haluamansa vaihtoehdon */
....
scanf ("%i", &choice);
/* lisäksi tarkistetaan, että valinta on sallittu arvo */
...
if (choice ==0) break;
printf("Enter x: "); scanf("%lg", &x);
fp = function[choice];
fx = fp(x);
printf("\n (%g) = %g\n", x, fx);
}
}

```

typedef double (\*funcptr) (double);

function	[0]	[1]	[2]	[3]	[4]	[5]	[6]
	↓	↓	↓	↓	↓	↓	↓
	NULL	sin	cos	tan	log	log_2	exp

## Kurssikokeesta

- n 2.5. klo 16-19
  - n Ylimääräinen koetilaisuus 8.5. klo 9-12 niille, jotka hyvästä syystä ovat estyneet osallistumasta 2.5. pidettävään kokeeseen
    - n Ilmoita Liisalle ([liisa.marttinen@cs.helsinki.fi](mailto:liisa.marttinen@cs.helsinki.fi)), jos olet tulossa tähän kokeeseen!
  - n Näihin kokeisiin saa osallistua vain, jos on jättänyt harjoitustyönsä tarkastettavaksi 24.4. mennessä
- n Kokeessa saa olla mukana A4:n kokoinen lunttilappu

## Vähän kokeesta: mitä pitää osata

- n Aiempien kurssin asioista perusohjelmointi, algoritmien kirjoittaminen, tietorakenteiden käyttö (taulukko, lista, pino) ja niiden tavanomaiset käsittelyrutiinit (lisääminen, poisto ja järjestäminen)
- n C-kielen syntaksi ja semantiikka. Kirjastorutiinien käyttöä ei sinänsä edellytetä, mutta esimerkiksi merkkijonojen ja tiedostojen käsittelyyn käytettävät tavanomaiset funktiot on syytä hallita.
- n Kielen rakenteista on hyvä hallita ainakin:
  - n Funktioiden ja niiden parametrien käyttö
  - n Taulukko
  - n Tekstitiedosto
  - n Linkitetty tietorakenne ja osoittimet
  - n Komentoriviparametrit
  - n Merkkijonot

## Lisää kokeesta

- n Näistä osattavista asioista muodostetaan sitten kokeessa erilaisia yhdistelmiä eri tehtävissä.
- n Esimerkiksi toukokuun 2005 kokeessa oli
  - n tehtävässä 1:
    - n funktion käyttöä ja tietojen lukemista käyttäjältä (eli tiedostosta stdin),
  - n tehtävässä 2:
    - n funktio, osoittimia ja linkitetyn listan kopiointi ja järjestäminen
  - n tehtävässä 3:
    - n Tekstitiedosto, komentoriviparametri ja taulukko
- n Koska koeaikaa on vain 2,5 tuntia, niin kolmeen tehtävään vastaaminen edellyttää jonkinlaista rutiinia C-ohjelmien kirjoittamisessa
  - n Tätä rutiinia on saatu jo Viopen harjoituksissa, mutta etenkin kurssin harjoitustehtäviä tehdessä!
  - n Lunttilappu helpottaa asioiden muistamista!