DTEK2003 Järjestelmäohjelmointi C++-kielellä 5 op

DTEK2003 Perustiedot 2011

- Luennoitsija: TkT Jerker Björkqvist
  - Åbo Akdemi, ICT Talo, huone B4052
    - jerker.bjorkqvist@abo.fi
- Luennot, 24.10.2011 – 16.12.2011
  - Keskiviikkoisin 12-14, Lambda
  - Perjantaisin 10-12, Alpha
  - `https://xprog28.cs.abo.fi/ro.nsf/W/cprogf`
- Tentit;
  - 19.12.2011,
  - 30.1.2012
  - 5.3.2012
  - Ilmoittautuminen: nettiopsu

DTEK2003 Perustiedot 2011

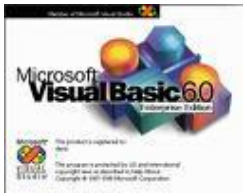
- Harjoitukset, huone A2065, 3.11.2011 – 9.12.2011
  - Torstaisin 8-12
  - Perjantaisin 12-14
  - 6 kertaa
    - 4 x setti harjoituksia
      - ville.utu.fi
    - 2 x kertaa laajempi tehtävä
      - moodle.utu.fi (tarvittava koodi: c++s11)
- Tenttioikeus:
  - Ville-tehtävät minimi 60% suoritettu
  - Ensimmäinen laajempi teht. suoritettu (ja hyväksytty)
- Arviointi / hyväksyminen
  - Kirjallinen tentti
    - (ville-tehtävät suoritettu min 60%)
  - Molemmat laajemmat tehtävät suoritettu

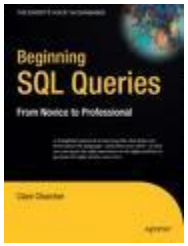
- Kaikki harjoitusasiat hoitaa: **Peter Larsson (ulpela@utu.fi)**

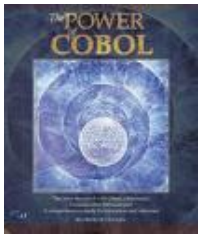
Kirjat / resurssit

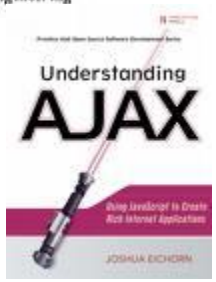
- C:
  - Kernighan&Ritchie: The C programming language
- C++:
  - Frank Brokken: C++ Annotations
    - **`http://www.icce.rug.nl/documents/cplusplus/`**
  - Stephen Prata: C++ Primer Plus
    - **`www.cplusplus.com`**
- C++ ympäristö (kirjastot + GUI)
  - Qt (Cross-Platform GUI toolkit)
    - **`qt.nokia.com`**
    - Cross-platform
    - Useampi ohjelmointikieli
  - wxWidget (aikaisemmin)
    - www.wxwidgets.org

Miksi opetella C/C++ ?





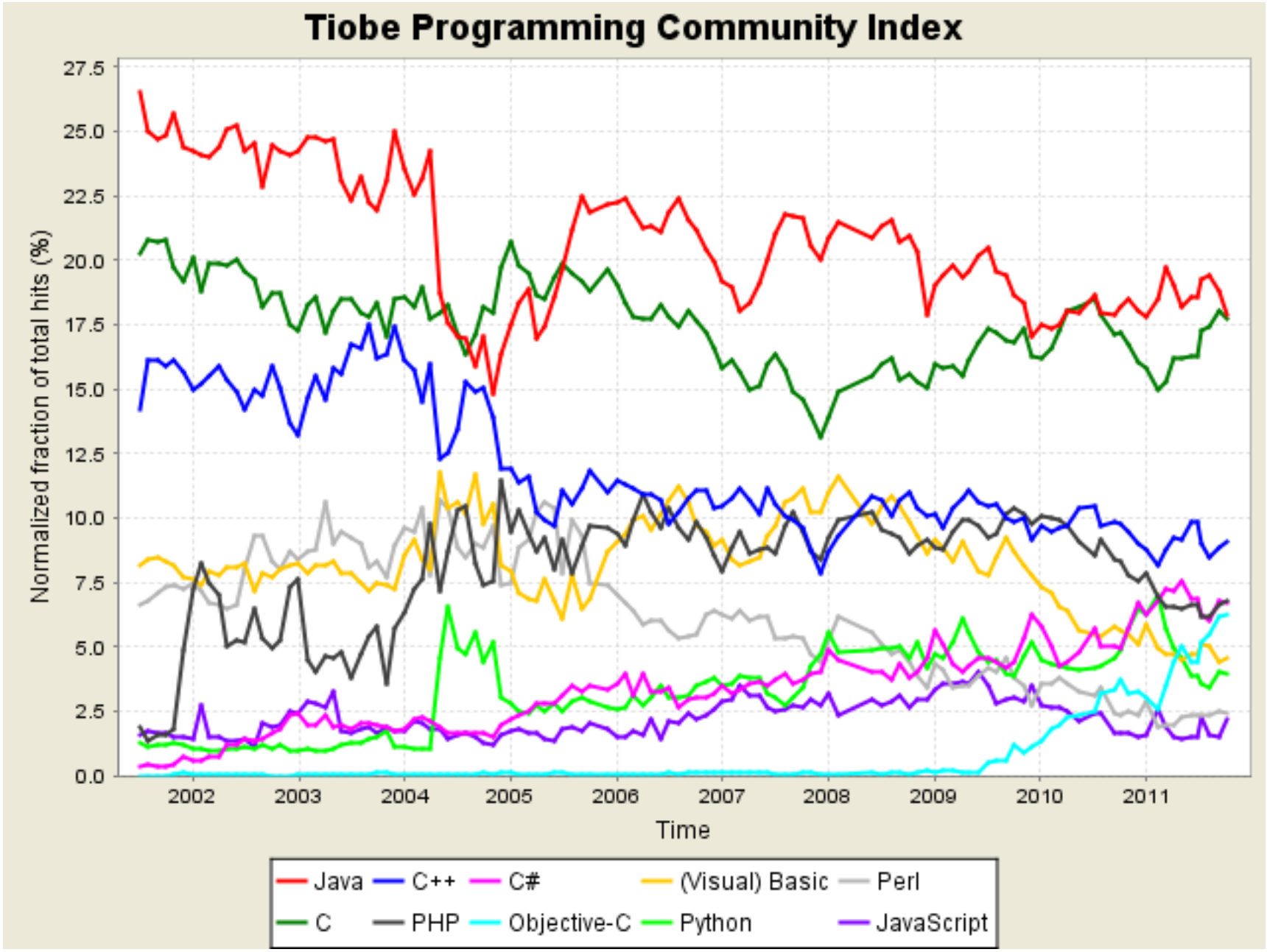




Position Oct 2011	Position Oct 2010	Delta in Position	Programming Language	Ratings Oct 2011	Delta Oct 2010	Status
1	1	=	Java	17.913%	-0.25%	A
2	2	=	C	17.707%	+0.53%	A
3	3	=	C++	9.072%	-0.73%	A
4	4	=	PHP	6.818%	-1.51%	A
5	6	↑	C#	6.723%	+1.76%	A
6	8	↑↑	Objective-C	6.245%	+2.54%	A
7	5	↓↓	(Visual) Basic	4.549%	-1.10%	A
8	7	↓	Python	3.944%	-0.92%	A
9	9	=	Perl	2.432%	+0.12%	A
10	11	↑	JavaScript	2.191%	+0.53%	A
11	10	↓	Ruby	1.526%	-0.41%	A
12	12	=	Delphi/Object Pascal	1.104%	-0.45%	A
13	13	=	Lisp	1.031%	-0.05%	A
14	14	=	Transact-SQL	0.909%	+0.09%	A
15	23	↑↑↑↑↑↑↑↑	PL/SQL			
				0.721%	-0.05%	A
	-	=	Assembly*	0.622%	-	B
20	17	↓↓↓	Ada	0.609%	-0.09%	B

Java lost almost 1% of its popularity in September 2011. If this trend continues, C will be number one again next month. Other interesting observations are that both Objective-C and Transact-SQL scored new all time highs, Assembly reentered the top 20, Visual Basic.NET gained many places (from #39 to #25), while F# had a hard time (from #23 to #46).

Source: http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html



Source: http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

C - käyttöalueet

- Järjestelmäohjelmointi
  - Käyttöjärjestelmät:
    - Ydin
    - Ajurit
  - Tietokantamoottrit
  - Middleware eri laajuudessa (myös C++)
- Sulautetut järjestelmät
  - DSP
  - Muut pienet tietokonejärjestelmät
- Yleensä missä
  - Suorituskyky on tärkeämpi kun tuottavuus
  - Halutaan kontrolloida tarkasti lopputulosta

Joitakin järjestelmiä C-kielellä

Käyttöjärjestelmien ytimet:

- Linux, Windows 7, MacOS, RTOS (uC/OS-II), RTEMS

Skriptikielien tulkit

- Python, Perl, PHP, Ruby,...

Sulautetut järjestelmät

- ECU:s (Elektroniset ohjausjärjestelmät)
- DSP (Digitaaliset signaaliprosessorit)
- Mikrokontrollerit
- .... Pitkä lista...

C - Lyhyt historia

- UNIX:n ensimmäiset versiot kirjoitettiin assemblerilla
  - Liian työlästä
- Tyypitön kieli BCPL (Richards) 1960-luvun keskivaiheilla
- Ken Thompson kehitti BCPL kielen perusteella järjestelmäohjemointia varten B-kielen
- Dennis Ritchie kehitti B, nimeksi tuli C, 1970-luvun alussa → tietotyyppien lisääminen
- 1978: Brian Kernighan, Dennis Ritchie kirja: K&R C: De facto standardi
- 1983: ANSI aloitti C-kielen standardointia, 1989 syntyi ANSI C89
- 1999: Uusi versio C99
- 2011: Draft C1X standardi (ei vielä valmis)

http://www.ixonos.com/easydata/customers/ixonos/files/OuluLinuxPresentations/Linux_seminar_stroustrup.pdf

C-kielen etuja

- Suora muistinkäyttö helppoa – myös muiden laitekomponenttien käyttö helppa
- Käyttöjärjstelmät / peruskirjastot C-kielellä, suoraviivaista kutsua C-ohjelmasta
- C-kirjastojen laajuus
- On olemassa monta hyviä C-kääntäjiä, jotka generoi tehokasta koodia
- On saatavilla melkein kaikille alustoille (mikrokontrollerista superkoneeseen)
- On luultavasti käytettävissä monta vuotta eteenpäin
- Monet muut kielet perustuvat C-kieleen (tai syntaksiin); C++, Java, C#, Objective-C, Javascript, PHP,...

C kielen huonoja puolia

- Liian helppoa tehdä katastrofaalisia ohjelmointivirheitä
- C-ohjelmien virheenetsintä on usein vaikeaa
- Matalan tason kieli: vaikeaa hallita suuria ohjelmakokonaisuuksia
- Houkutus oikoteille (hacks) on suuri, joka johtaa vaikeuksiin myöhemmin

Asiaan - HelloWorld C-kielellä

```
/* helloword.c */
#include <stdio.h>

int main(void) {
   printf("Hello, world!\n");
   return 0;
}
```

Kääntäminen ja ajo

```
% gcc helloworld.c -o helloworld
% ./helloworld
```

C-kielen järjestelmäläheisyys

- Helloworld.c ajo:

```
main() {
        printf(”Hello World\n”);
}
```

  - 25 järjestelmäkutsua (kutsuja ytimeen), `strace` -käyttäen

- Vastaava HelloWorld.java ajo

```
public class HelloWorld {
    public static void main(String args[])
{
        System.out.println("Hello
World!");
    }
}
```

  - 2248 järjestelmäkutsua !!

C-kielen järjestelmäläheisyys (2)

```
$> strace ./helloWorld

execve("./helloWorld", ["./a.out"], [/* 24 vars */]) = 0
uname({sys="Linux", node="borken", ...}) = 0
brk(0)                                  = 0x804962c
old_mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x40016000
open("/etc/ld.so.preload", O_RDONLY)    = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY)      = 3
fstat64(3, {st_mode=S_IFREG|0644, st_size=82445, ...}) = 0
old_mmap(NULL, 82445, PROT_READ, MAP_PRIVATE, 3, 0) = 0x40017000
close(3)                                = 0
open("/lib/libc.so.6", O_RDONLY)        = 3
read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0 \304\1"..., 1024) = 1024
fstat64(3, {st_mode=S_IFREG|0755, st_size=5735106, ...}) = 0
old_mmap(NULL, 1267176, PROT_READ|PROT_EXEC, MAP_PRIVATE, 3, 0) = 0x4002c000
mprotect(0x40158000, 38376, PROT_NONE)  = 0
old_mmap(0x40158000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED, 3, 0x12b000) =
0x40158000
old_mmap(0x4015e000, 13800, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS,
-1, 0) = 0x4015e000
close(3)                                = 0
munmap(0x40017000, 82445)               = 0
fstat64(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 3), ...}) = 0
old_mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x40017000
write(1, "Hello world\n", 12Hello world
)           = 12
munmap(0x40017000, 4096)                = 0
_exit(0)                                = ?
```

C perusteet

- Rakenteeltaan Pascal tai Fortran-tyyppinen (proceduraalinen kieli)
- Muuttujia käytetään datan varastointiin
- Ohjelmarakenne luodaan käyttämällä ja kutsumalla funktioita
- Ohjelman ajo kontrolloidaan käyttämäälla silmukoita, ehtolauseita (if) ja kutsumalla funktioita
- Pointterien (epäsuora osoitus) käyttö keskeistä (mutta tämä myöhemmin)

```
/* test.c */
#include <stdio.h>
void test(int a);
int main(void) {
int a=23;
int *b = &a;
test(a);
}
void test(int a) {
if (a>54) {
printf("parameter for test() is
larger than 54\n");
}
}
```

C – kieli - tiedostot

- Lähdetiedostot (.c)
  - Sisältää funktioiden implementaatiot
  - Ohjelma jaetaan mieluiten useampaan lähdetiedostoon
- Header-tiedosto (.h)
  - Funktioiden prototyypit, datatyyppien määrittelyt
  - Prototyypit ja määrittelyt käytetään (sisällettytään lähdekoodiin) käyttämällä #include-lauseella lähdekoodissa

Prototyyppi: Funktioiden deklaraatiot, mukaanlukien parametrit ja niiden tyypit

```
int myfunc(float a, int b);
```

C-ohjelman perusrakenne

- Ohjeet esikääntäjälle :`#include <stdio.h>`
- Muuttujien deklaraatiot: `int a;`
- Funktioprototyypit: `int main(void);`
- Funktioiden määritelmät (implementaatio)

```
int main(void) {
....
}
```

- C on case-sensitive
- Muuttujia pitää deklaroide ennen kuin niitä voi käyttää
- Funktioita voi kutsua ilman että vastaava prototyyppi on käytettävissä (muttei suositella)
- Kommentit: `/* ... */` (ANSI C99 myös `//`)

Tietotyypit - kokonaisluvut

- `char` – min 1 tavu
- `short` – min 2 tavu
- `int` – min 2 tavua, yleensä 4 tavua
  - Mutta 64-bittinen arkitehtuuri: 8 tavua
- `long` – min 4 tavua
- `long long` – min 8 tavua
- Tyyppien numeroavaruudet: limits.h
- Vakiot tekstissä
  - `2, 6L, 88LL, 0x20, 0775`

C-koodisaa voidaan tarkentaa mikä datatyyppi halutaan että C-kääntäjä pitäisi muodstaa (eli ASCII-tekstin jäsentelyn yhteydessä)
Esim: L – long, LL - long long, 0x – hexadecimal ....

Tietotyypit - liukuluvut

- `float` – yleensä 4 tavua → $10^{37,}$ 6 numeron tarkkuus
- `double` – min kuten float
- `long double` – min kuten double
- Vakiot tekstissä
  - `3.14, 0.5e-7, 12.4f, 145.67L`
- Numeroavaruudet `float.h`

Datatyypit - boolean

- C-kielessä ei ole erillisiä totuusarvomuuttujia (mutta ANSI C99 toisaalta on)
- Kokonausluvut toimivat totuusarvoina → 0 on false, muut true

```
a=99;
if (a) do_something();
```

Taulukot

- Taulukko: yksi ulottuvuus

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    int lotto[7]; /* 7 numeroa per rivi*/
    for (i=0; i<7; i++)
      lotto[i] = rand() % 37;
    return 0;
}
```

- Useampi ulottuvuus

```
#include <stdio.h>
int lottolomake[10][7];
lottolomake[2][5] = rand() % 37;
```

Taulukot

- Taulukko pitää alustaa ennen käyttöä

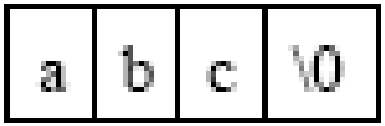
```
#include <stdio.h>
#include <stdlib.h>
int main() {
   int lotto[7];
   printf("%d\n", lotto[5]); /* undefined */
   printf("%d\n", lotto[33]); /* typical buffer
   overrun, varying result: Undefined /
   segmentation fault / general protection fault
   */
}
```

HUOM: Kääntäjän näkökulmasta kaikki OK!

Merkkijonot

- C-kielessä ei ole erityistä merkkijonotyyppiä. Merkkijono on vaan tavallinen taulukko, jossa merkkijonon päättää ’0’-merkki

```
char a[] = "abc";
a[0] = ’a’
a[1] = ’b’
a[2] = ’c’
a[3] = ’\0’
```



- Merkkijonoja tuetaan LIB-C kirjastossa, josta löytyy useampi funktio. header-tiedostossa `string.h` löytyy funktioiden prototyypit

Tyyppimuunnokset

```
#include <stdio.h>
int main(void)
{
   int i,j = 12;       /* only j is initialized*/
   float f1,f2 = 1.2;

   i = (int) f2;       /* explicit: i <- 1, 0.2 lost */
   f1 = i;             /* implicit: f1 <- 1.0 */

   f1 = f2 + (float) j; /* explicit: f1 <- 1.2 + 12.0 */
   f1 = f2 + j;        /* implicit: f1 <- 1.2 + 12.0 */
}
```

Tyyppimuunnokset

- Yllätykset / väärin ajateltu??

```
   int a, b=1;
   a = 2 * (b / 2);  /* a has now value 0 */

MUTTA:
   a = 2 * (b / 2.0); /* a has now value 1 */

-----------------
   int sum=7, count=3;
   float average;

   average = sum/count; /* gives average  2.0 */
MUTTA:
   average = (float)sum/count; /* gives average 2.3333 */
```

Operaattorit

- Aritmeettiset
  - `int i = i+1; i++; i--; i *= 2;`
  - `+, -, *, /, %`
- Relaatiot, logiikka
  - `<, >, <=, >=, ==, !=`
  - `&&, ||, &, |, !`
- Bittioperaatiot
  - `& - and, | - or, ~ - complement`
  - `<<,>> - left-shift, right-shift`
  - `^ - xor`
- HUOM
  - `= arvon asetus`
  - `== vertailu`

Rakenteet

- **`if (expr) statement`**
  - Jos **`expr`** arvo on eri kun 0 niin **`statement`** suoritetaan
- **`if (expr) statement else statement2`**
  - Jos **`expr`** arvo on 0 niin **`statement2`** **suoritetaan**

`statement` on joko yksinkertainen: **`a=2;`**, tai yhdistetty: **`{a=2; b=1;}`**

`if` - esimerkki

```
/*
if.c
*/
#include <stdio.h>
int main(void)
{
   int a = 2;
   int b = 3;
   if(a > b) printf("a on suurempi kuin b!\n");
   else printf("a ei ole suurempi kuin b!\n");
   return 0;
}
```

while

- **`while (expr) statement`**
  - **`statement`** suoritetaan niin kauan kun **`expr`** arvo on eri kuin 0
- **`do statement while (expr)`**
  - Eri versio samasta rakenteesta

for

- **`for (pre_expr; loop_expr; post_expr) statement`**
- Silmukkarakenne
  - Ensin suoritetaan **`pre_expr`**
  - Niin kauan kun **`loop_expr`** on eri kuin 0, suoritetaan ensin **`statement`** ja sitten **`post_expr`**
- Tehokas rakenne
  - Melkein kaiken tyyppiset silmukat voidaan tehdä **`for`** avulla

for - esimerkki

```
int a,i,n=5;
a=0;
for (i=0; i<n; i++) a += i*i;
```

Vastaa

```
int a,i,n=5;
for (a=0,i=0; i<n; a += i*i++) ;
```

Switch - rakenne

```
switch (expr) {
  case constant-expression :
       statement
     break
   case constant-expression:
    statements
    break;
   default:
}
```

Funktiot

- Ohjelma jaetaan pienempiin osiin funktioiden avulla
  - Modulaarisuus – edesauttaa
    - Koodausta, debuggausta
  - Koodin uusiokäyttö

- Funktion deklaraatio – prototyyppi
- Funktion määrittely – prototyyppi + koodi = implementaatio

Funktiomäärittelyt

```
type name(parameter_list){
    declarations
    statements
 }
```

- Jos funktion ei palauta arvoa, tyypiksi annetaan **`void`**
- Parametrilista koostu tyypistä + muuttujanimestä **`(int a, float b)`**
- Funktiomäärittelyssä **`a`** ja **`b`** on formaaleja parametreja
- Funktiokutsu **`f(x,4)`** yhteydssä **`x`** ja **`4`** ovat todelliset parametri, eli argumentitent

Funktiokutsu

- Ennen kuin funktio kutsutaan koodissa, suositellaan että funktion prototyyppi on tiedossa (eli kääntäjä tuntee prototyyppiä)
  - Paluuarvon tyyppi oikea
  - Parametrit voidaan validoida
- Yleensä käytetään header-tiedosto, eli

```
#include <stdio.h>
```

- Funktiokutsu = funktion nimi + parametrit suluissa **`pow(2,4)`**
- Funktiokutsut tapahtuu aina call-by-value
  - Tästä voidaan olla eri mieltä…

Funktiota - esimerkkejä

```
#include <stdio.h>
int sum(int a, int b);
      /* function prototype at start of file */

void main(void){
   int total = sum(4,5); /* call to the function
   */

printf("The sum of 4 and 5 is %d", total);
}

int sum(int a, int b){   /* the function itself
                    - arguments passed by value*/
   return (a+b);       /* return by value */
}
```

`main()` -funktio

- `main()` – funktiosta normaalin ohjelman suoritus alkaa
  - Tarvittava startup-koodi linkataan känntämis/linkkausvaihessa niin, että tietyn alustuksen jälkeen kutsutaan main()-funktiota
- Argumentit: `main (int argc, char *argv[])`
  - `argc` –montako parametrejä on annettu ohjelman käynnistäessä
  - `argv[]` – pointteri argumenttilistaan
    - `argv[0]` – ohjelmanimi
    - `argv[1]` – ensimmäinen argumentti

Muuttjat - näkyvyysalue

- Oletus C-kielessä on että muuttujien näkyvyys on rajoittunut tiedostoon
  - Funktion ulkopuolella deklaroitu muuttuja on *globaali* käännöslohkon sisällä
  - Funktion sisällä deklaroitu muuttuja on *lokaali* funktion sisällä
  - Rakenteessa (**`for, while, if`**) deklaroitu muuttuja on *lokaali* rakenteen sisällä
  - Muuttuja joka deklaroidaan funktion sisällä on lokaali funktion sisällä

Näkyvyysalue - esimerkki

```
#include <stdio.h>

int gCommonPar; /* Globaali muuttuja */

void main(void){
   int i=0; /* Lokaali funktion sisällä */
   for (i=0; i<5;i++) {
       int j; /* Lokaali rakenteen sisällä */
       j=i*5+gCommonPar;
       printf("Laskutoimituksen tulos: %i\n", j);
   }
   j=3; /* Käännösvirhe, muuttuja j ei tässä
   rakenteessa */
}
```

Muuttuja – elinikä/tallennus

- **`static, auto`** määrittää muuttujien elinikä (ja tallennus)
  - **`static`** – muuttuja sijoittuu heap:iin, saa oman paikkansa muistiavaruudessa, ja on olemassa ohjelman suorituksen alusta loppuun
  - **`auto`** – muuttuja sijoitetaan pinoon (stack) muistipaikka on käytössä vaan niin kauan kuin ollaan kyseisen funktion sisällä
  - **`auto`** on lokaalien muuttujien oletus – **`static`** on globaalien muuttujien oletus

Elinikä / esimerkki

```
#include <stdio.h>

int gimmenext() {
  static int next=0; /* Heap */
  return ++next;
}
int gimmenext2() {
  int next=0; /* Auto(stack) */
  return ++next;
}
int main() {
   printf("%i\n",gimmenext());
  printf("%i\n",gimmenext());
   printf("%i\n",gimmenext2());
  printf("%i\n",gimmenext2());

}
```

```
% ./gimmenext
1
2

1
1
```

Muut tallenusluokat

- **`register`** – ohje kääntäjälle sijoittaa muuttujan, jos vaan mahdollista, processorin rekisteriin
- **`volatile`** – ohje kääntäjälle jättää optimoinnit liittyen tähän muuttujaan tekemättä
  - esim muuttujia jotka suoraan osoittavat rautaan
- **`extern`** – muuttuja on saanut säilytystilaa toisen kännöslohkon kautta, eli ohje kääntäjälle käyttää tätä muualla deklroitua muuttujaa
- **`const`** – muuttuja on määritelty vakioksi, ei kääntäjä ei salli muutoksia muuttujan arvoon

Esimerkki - **extern**

```
/* fil_a.c */

int g_CommonParameter = 5;
```

```
/* fil_b.c */
#include <stdio.h>
extern int g_CommonParameter;
int main() {
  printf("Parameter=%i\n", g_CommonParameter);
}
```

```
% gcc fil_a.c fil_b.c -o prog
% ./prog
Parameter=5
```

Struktuurit - tietueet

- C:ssä voidaan implementoidaan tietueita

```
#include <string.h>
#define MAX_STRING 30

struct t_person {
   char name[MAX_STRING];
   int age;
   int shoesize;
};
int main() {
   struct t_person Person; /* Tietue pinossa */
   Person.age = 23;  /* Tietueen kenttä
                              ”.”-merkin avulla*/
   strcpy(Person.name, “kalle”);
}
```

Tietueet tietueen sisällä

```
#define MAX_PARTICIPANTS 100
#define MAX_STRING 30
struct t_course {
   char name[MAX_STRING];
   int nPersons;
   struct t_person Persons[MAX_PARTICIPANTS];
};

main() {
   struct t_course Course;
   strcpy(Course.name, "Programming C++");
   Course.nPersons = 1;
   Course.Persons[0].age = 77;
   strcpy(Course.Persons[0].name, "Axel Eklund");
}
```

Datatyypin synonyymi

```
typedef unsigned short WORD;

WORD wIndex; /* sama kuin ”unsgined short wIndex” */

typedef struct t_person Person;

Person Jag;    /* struct t_person Jag; */

typedef struct t_point {
   WORD x;
   WORD y;
} Point;

Point point;
point.x = 12; point.y=44;
```

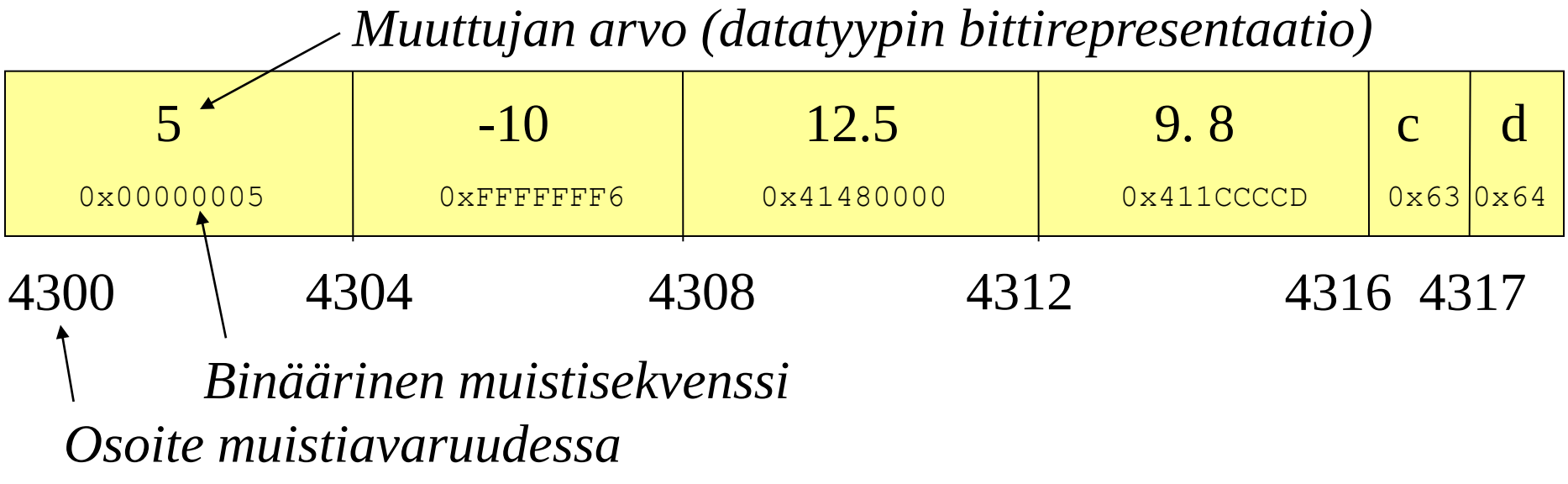
Esimerkki

```
#include <stdio.h>

int main(void)
{
   int nstudents = 0; /* Initialisering */

   printf("How many students does ÅA have ?:");
   scanf ("%d", &nstudents);  /* Läs input */
   printf("ÅA has %d students.\n", nstudents);

   return 0;
}
$How many students does ÅA have ?: 5000 (enter)
ÅA has 5000 students.
$
```

Muuttujat muistissa / osoitteet

Muuttjalla on aina vastaavuus
Fyysisessa muistissa
Datatyyppi / tietty arvo
→ Tietty bittisekvenns muistissa

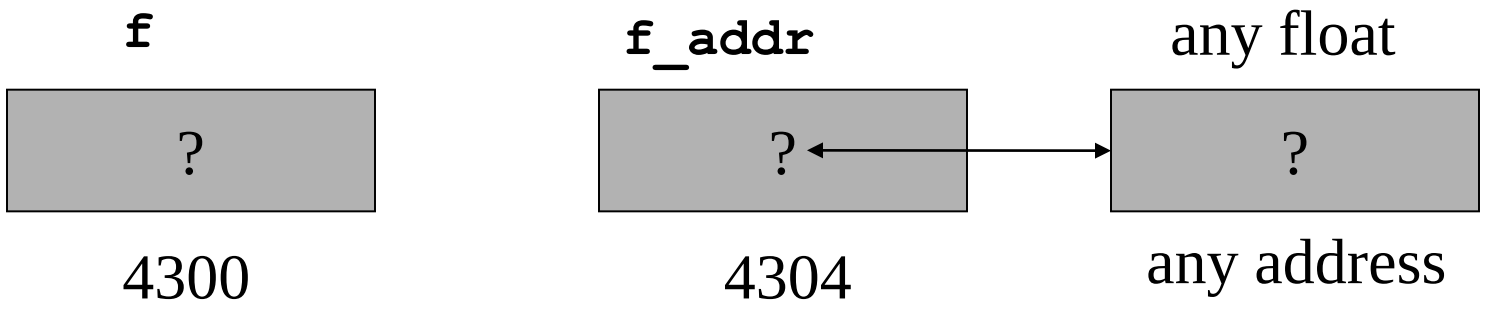
```
int x = 5, y = -10;
float f = 12.5, g = 9.8;
char c = 'c', d = 'd';
```



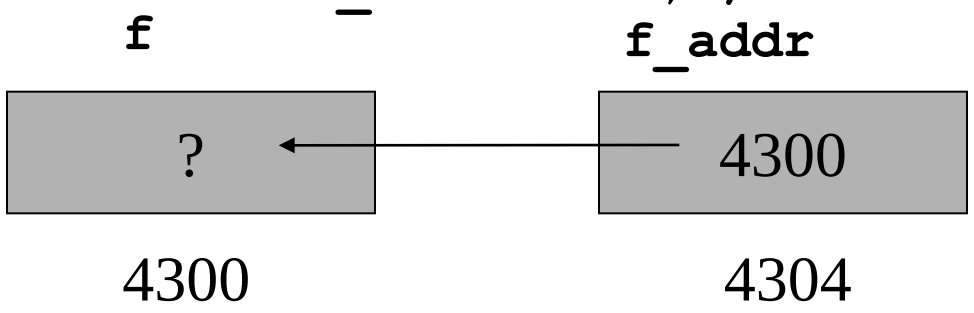
Osoittimet

- *Pointteri/osoitin*= muuttja, jonka arvo osoittaa tiettyyn paikkaan muistissa

```
float f;          /* datamuuttuja */
float *f_addr;    /* osoitinmuuttuja */
```




```
f_addr = &f; /* & = osoiteoperaattori */
```

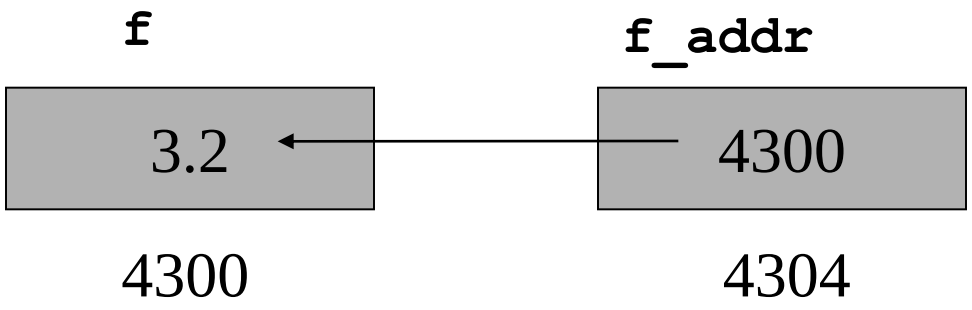


Osoittimet / esimerkki

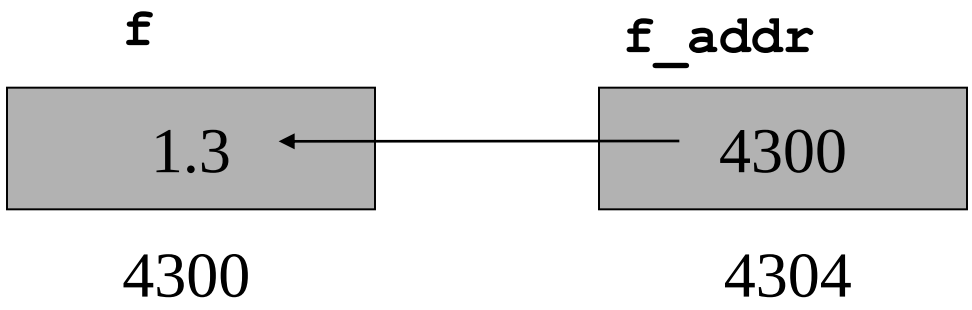
```
#include <stdio.h>

void main(void) {
int j;       /* j = <undefined> */
int *ptr;    /* ptr = <undefined> */

ptr=&j;    /* alusta ennen käyttöä */
           /* *ptr=4 ei alusta osoitinta */

*ptr=4;      /* j <- 4 */

j=*ptr;      /* j <- ??? */
}
```

Osoittimet 2

```
*f_addr = 3.2;     /* epäsuora osoitus*/
```

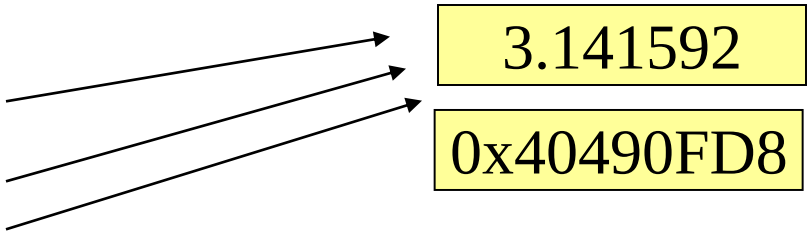



```
float g=*f_addr; /* indirektion:g on nyt 3.2 */

f = 1.3;
```



Osoittimet – datatyypin vaihto

```
#include <stdio.h>

int main(void) {
float f = 3.141592;
void *ptr;  /* yleinen osoitin, ei tyyppiä */
float *f_ptr;
int *i_ptr;

ptr = (void *)&f;
f_ptr = &f;
i_ptr = (int *)&f

printf("Val: %f, adress: %p, : int %i\n",
      *f_ptr, ptr, *i_ptr);
}
% Val: 3.141592, adress: 0xbf82e238, int: 1078530008
```



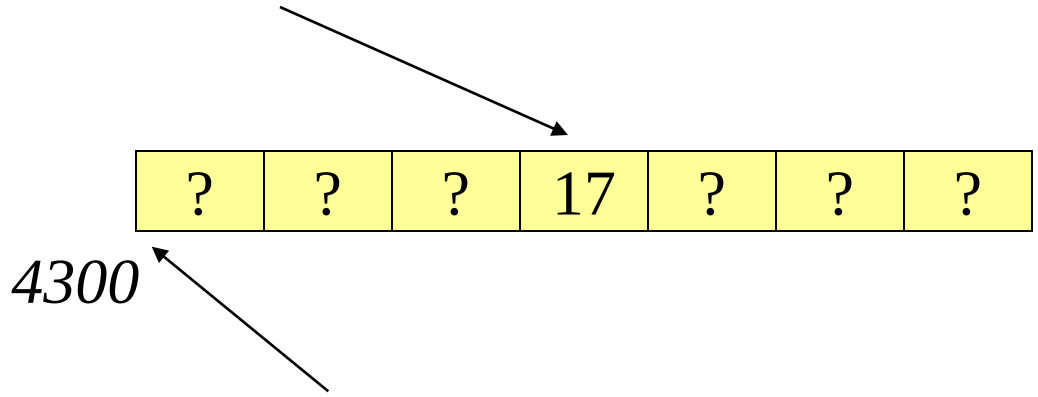
sizeof()-operaattori

- **`sizeof(x)`** – anta tietotyypin x koko (bytes)
- esim (oletetaan 32 bittinien arkkitehtuuri)
  - **`sizeof(int) = 4`**
  - **`sizeof(float) = 4`**
  - **`sizeof(int *) = 4`**
  - **`sizeof(Person) = 40`**

    `char[30] + int + int + alignment`
  - **`sizeof(char a[20]) = 20`**

Taulukot uudestaan

```
int main(void) {
   int lotto[7];

   lotto[3] = 17;
}
```



muuttuja lotto on käytännössä osoitin, ja osoittaa taulukon ensimmäiseen elementtiin

lotto[3] voidaan laskea myös

```
   *(lotto + 3 * sizeof(int))
```

Taulukon osoittimet

Kompilaattori tekee seuraavat laskutoimitukset

```
&(lotto[3]) = lotto + sizeof(integer)*3
```

Taulukko voidaan myös accessoida seuraavasti:

```
*(lotto+3) = 17 /* vastaa lotto[3]=17 */
```

Mutta:

```
*lotto+3 = <int>
```

Taulukkoa voidaan myös käyttää normaalin osoittimen kautta:

```
int *lotto_ptr = lotto;
lotto_ptr[3] = 17;
```

Taulukot lisää

Mikä seuraavien ero?

```
int lotto[7];
  JA
int *lotto_ptr;
```

lotto : Vakio osoitin taulukkoon, samalla on tilaa data varteen varattu. Osoittimien arvo ai voit muuttaa.

lotto_ptr: Muuttuja, jonka arvo voidaan muuttaa.

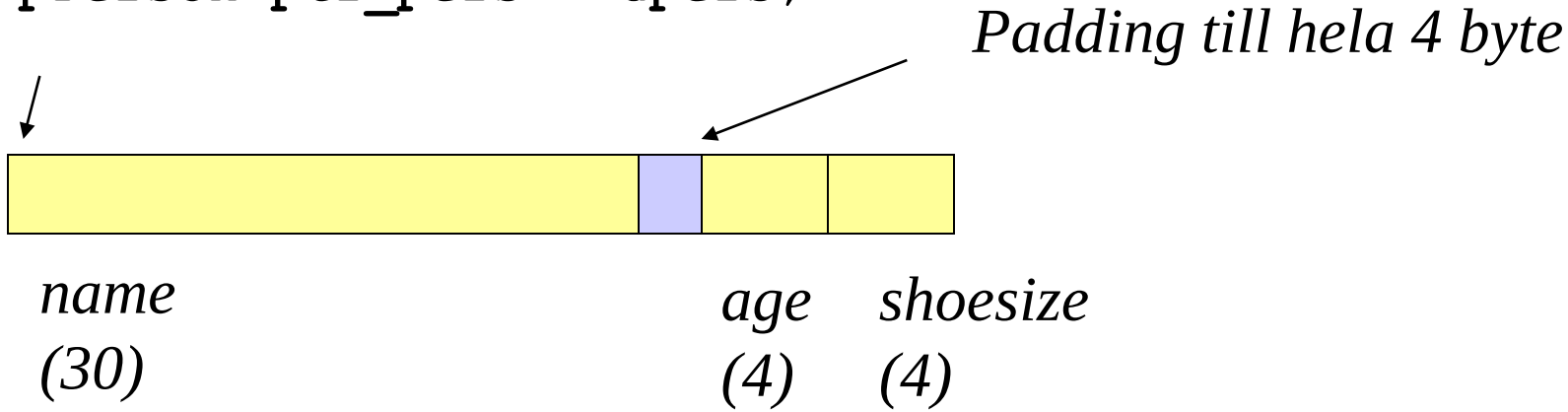
Lisää osoittimia

```
int month[12]; /* month is a pointer to base address 430*/
int *ptr;

month[3] = 7;  /* month address + 3 * int elements
        => int at address (430+3*4) is now 7 */

ptr = month + 2; /* ptr points to month[2],
        => ptr is now (430+2 * int elements)= 438 */
ptr[5] = 12;
         /* ptr address + 5 int elements
              => int at address (438+5*4) is now 12.
            Thus, month[7] is now 12 */

ptr++;           /* ptr <- 438 + 1 * size of int = 442 */
(ptr + 4)[2] = 12;   /* accessing ptr[6] i.e., month[9] */
```

- Nyt `month[6]`, `*(month+6)`, `(month+4)[2]`, `ptr[3]`, `*(ptr+3)` om itse asiassa sama muuttuja!!

Tietueosoittimet

```
typedef struct t_person {
   char name[30];
   int age;
   int shoesize;
} Person;

typedef Person * pPerson;

int main() {
   Person pers;
   pPerson ptr_pers = &pers;
}
```



By-value / by-reference

Sanottiin aikaisemmin että KAIKKI parametrit C-kielessä ovat by-value

Miten toimii esim. taulukot?

```
char str1[30] = "Hei", str2[30];
strcpy(str2, str1);
```

Arvo mikä siirretään funktioon on osoitinmuuttujan arvo, eli osoite missä merkkitaulukko sijaitsee muistissa. Eli osoitin siirretään ”*by-value*” (str1, str2 on (vakio)osoittimia), tämä voitaisiin kutsua myös *by-reference*

By-value / by-reference

C-kielessä ohjelmoijalla on aina kontrolli siihen muistiin mitä olet allokoinut:

→ Kutsutut funktiot eivät voi muuttaa allokoimasi muistia ilman ”lupaasi” (eli jotta funktio voisi muttaa toisen funktion sisäisiä muuttujia, se pitää saada käyttöönsä osoitin tähän muuttujaan (by reference))

By value / by reference

```
int inc(int a) { /* tämä funktio ei voi muuttuu kutsujan
                    käytössä oleva muuttujaa */
   return a+1;
}
void inc_ref(int *a) { /* tämä funktio on saanut
   käyttöönsä osoittimen alkuperäiseen muuttujaan, eli se
   voi muuttaa alkuperäistä muuttujaa
*/
   *a++;
}

int main() {
   int b=0;
   b = inc(b);    /* by value: muuttujan arvo */
   inc_ref(&b);  /* by ”value”: osoitin muuttujaan */
}
```

By value / by reference

```
#include <stdio.h>
void swap(int *, int *);

main() {
  int num1 = 5, num2 = 10;
  swap(&num1, &num2);
  printf("num1 = %d and num2 = %d\n", num1, num2);
}

void swap(int *n1, int *n2) { /* passed and returned by
                                 reference */
  int temp;

  temp = *n1;
  *n1 = *n2;
  *n2 = temp;
}
```

Taulukot parametreina

```
void init_array(int array[], int size) ;

int main(void) {
  int list[5];

  init_array(list, 5);
  for (i = 0; i < 5; i++)
    printf("next:%d", array[i]);
}

void init_array(int array[], int size) { /* miksi koko
   annetaan ? */
        /* taulukot siirretään "by-reference"- eli
      osoitin taulukon ensimmäiseen arvoon */
 int i;
 for (i = 0; i < size; i++)
    array[i] = 0;
}
```

Funktio-osoittimet

Osoitinmuuttuja voi myös osoittaa funktioon, eli funktio-osoitin

```
int func(); /*function returning integer*/
int *func(); /*function returning pointer to integer*/
int (*func)(); /*pointer to function returning integer*/
int *(*func)(); /*pointer to func returning ptr to int*/
```

- Fördel ? mera flexibilitet

Funktioosoitin - esimerkki

```
#include <stdio.h>

void myproc (int d);
void mycaller(void (* f)(int), int param);

void main(void) {
   myproc(10);              /* call myproc with parameter 10*/
   mycaller(myproc, 10); /* and do the same again ! */
}

void mycaller(void (* f)(int), int param){
  (*f)(param);       /* call function *f with param */
}

void myproc (int d){
       . . .              /* do something with d */
  printf("Value: %d\n", d);
}
```

Mihin funktio-osoittimet käytetään?

Esimerkki 1:

Funktion rekisteröinti toisessa ohjelmassa, mikä voi suorittaa kuva-käsittelyä (tyypillisesti plug-in)

Esimerkki 2:

Funktion rekisteröinti, mikä reagoi signaaleihin, signaalikäsittelijän käyttöönotto

```
#include <signal.h>
typedef void (*sighandler_t)(int);
sighandler_t signal(int signum, sighandler_t handler);
```

Vielä laajemmin

Tehtävä
Luo taulukko jossa N osoitinta funktioihin, jotka palauttavat osoittimet funktioihin jotka palauttavat osoittimet merkkeihin

```
Version 1. char *(*(*a[N])())();

Version 2. Build the declaration up in stages, using
typedefs:

typedef char *pc; /* pointer to char */
typedef pc fpc(); /* function returning pointer to char
*/
typedef fpc *pfpc; /* pointer to above */
typedef pfpc fpfpc(); /* function returning... */
typedef fpfpc *pfpfpc; /* pointer to... */
pfpfpc a[N]; /* array of... */
```