

T-106.3100, Ohjelmoinnin jatkokurssi T2

Luento 3:
Tyyppejä

Petri Ihantola

Tänään luvassa

- Tietotyyppejä
- Merkkijonoja

Ennalta määritellyt perustietotyypit

- perustyytit
 - char
 - int
 - float
 - double
- Lyhyt tai pitkä (int ja double)
 - short int
 - long int
 - long double
 - tämä ei kuitenkaan takaa mitään
- Etumerkillinen tai etumerkitön (esimerkkejä)
 - unsigned short int
 - signed char

tyyppien koon selvittäminen

```
#include <stdio.h>
#include <stddef.h> // sizeof
#include <limits.h> // rajat: char, int, short, long, rang
// <float.h> rajat: float, double

main() {
    short x;
    printf("int is size %d\n", sizeof (int));
    printf("x is size %d\n", sizeof x);
    printf("long int is size %d\n", sizeof (long));
    printf("min short int value is %d\n", SHRT_MIN);
}
```

Vakiot

- Symbolisiin vakioihin olemme jo tutustuneet
- `const double PI=3.14159;`
 - näkyvyysalue
 - tyyppi
- Const ja osoittimet:

```
/* http://en.wikipedia.org/wiki/Const [24.1.2007] */
void Foo( int      *      ptr,
          int const *      ptrToConst,
          int      * const constPtr,
          int const * const constPtrToConst )
{
    *ptr = 0; // OK: modifies the pointee
    ptr = 0; // OK: modifies the pointer
    *ptrToConst = 0; // Error! Cannot modify the pointee
    ptrToConst = 0; // OK: modifies the pointer
    *constPtr = 0; // OK: modifies the pointee
    constPtr = 0; // Error! Cannot modify the pointer
    *constPtrToConst = 0; // Error! Cannot modify the pointee
    constPtrToConst = 0; // Error! Cannot modify the pointer
}
```

perustietotyypit - enumeraatiot

```
enum days {Ma=1, Ti, Ke, To, Pe, La, Su};
```

```
typedef enum days Days; /* miksi tämä halutaisiin? */
```

```
...
```

```
Days d;
```

```
d=Ma;
```

```
d=Su; /* sama kuin d=7 */
```

```
enum Boolean { false=0, true}; /* =0 ei ole välttämätön? */
```

Tietueet - struct

Tapa 1:

```
struct student {  
    char name[11];  
    int id;  
}
```

```
struct student s; /* määrittelee muuttujan s */
```

Tapa 2:

```
struct {  
    char name[11];  
    int id;  
} s, *ps; /* määrittelee muuttujat s ja ps (osoitin) */
```

```
/* tämän jälkeen ei voi enää määritellä tyypille uusia  
muuttujia koska structilla ei ole nimeä (sen voisi antaa) */
```

Tietueet, typedef ja kenttien käyttäminen

Yleinen tapa tietueen määrittelyyn:

```
typedef struct student { /* Idiomi: typedef ja struct */  
    char name[11];  
    int study_year;  
} Student;
```

```
Student s;
```

```
s.study_year = 2;
```

Osoittimet tietueisiin

- Tietueiden käyttäminen funktioiden argumenttina ei ole yleensä suositeltavaa (miksi?)

Osoittimet tietueisiin

- Tietueiden käyttäminen funktioiden argumenttina ei ole yleensä suositeltavaa (miksi?)
 - tietue kopioidaan kokonaisuudessaan pinokehykseen
 - Osoittimien käyttö

```
typedef struct student {  
    char name[11];  
    int study_year;  
} Student;
```

```
Student *s;
```

```
s->study_year = 2; /* (*s).study_year = 2 */
```

Unioni

- Syntaksi hieman kuin tietueella, mutta tavoite tyystin eri
- Samaan muuttujaan voidaan säilöä erityyppistä tietoa

```
union mytypes {  
    int i;  
    float f;  
    char c;  
} u;
```

```
u.i=2;  
u.f=2,1; /* edellinen arvo menetetään */  
u.c='a'; /* edellinen arvo menetetään */
```

Unioni

- Miten hinta asetetaan?

```
struct {  
    char title[50];  
    char author[50];  
    union {  
        float dollars;  
        int yens;  
    } price;  
} book;
```

```
struct {  
    char title[50];  
    char author[50];  
    union {  
        float dollars;  
        int yens;  
    };  
} book;
```

Lisää määreitä muuttujille (tallennusluokat)

- register int i;
 - vinkki kääntäjälle, että kyseinen muuttuja kannattaa pyrkiä sijoittamaan rekisteriin
 - muuttujille joihin halutaan nopea pääsy
 - optimointi kannattaa ehkä kuitenkin jättää kääntäjälle
 - &-operaattoria ei voi käyttää
 - rekisterien koko rajoittaa mahdollisten muuttujien tyyppejä

Lisää määreitä muuttujille (tallennusluokat)

- `static int globaali_muuttuja;`
`int toinen_globaali;`
`int main() {...`
 - Oletusmääre globaaleille muuttujille
 - näkyvyys vain kyseisessä tiedostossa
- `int main() {`
`static int muuttuja = 1;`
 - alustetaan käännösaikana
 - säilyttää arvonsa kutsukerrasta toiseen

Lisää määreitä muuttujille (tallennusluokat)

- ```
static int globaali_muuttuja;
int toinen_globaali;
int main() {...
```

  - Oletusmääre globaaleille muuttujille
  - näkyvyys vain kyseisessä tiedostossa
- ```
int main() {  
    static int muuttuja = 1;  
    ...  
}
```

 - alustetaan käännösaikana
 - säilyttää arvonsa kutsukerrasta toiseen
- ```
char *foo() {
 char string[] = "merkkijono";
 return(string);
}
```

# Lisää määreitä muuttujille (tallennusluokat)

- ```
static int globaali_muuttuja;  
int toinen_globaali;  
int main() {...
```

 - Oletusmääre globaaleille muuttujille
 - näkyvyys vain kyseisessä tiedostossa
- ```
int main() {
 static int muuttuja = 1;
 ...
}
```

  - alustetaan käännösaikana
  - säilyttää arvonsa kutsukerrasta toiseen
- ```
char *foo() {  
    static char string[] = "merkkijono";  
    return(string);  
}
```

 - ei enää muistivirhettä, koska string-muuttujaa ei varata pinosta

Lisää määreitä muuttujille (tallennusluokat)

- Tiedosto1:
int globaaliTaulukko[KOKO];

Tiedosto2:
extern int globaaliTaulukko[]

- Mahdollistaa globaaleiden muuttujien käyttämisen eri moduleissa
- extern kertoo, että muuttujalle on varattu tilaa jossakin toisaalla, joten sille ei tarvitse varata tilaa
- Määrittele globaali muuttuja vain kerran ja käyttä muualla extern määrettä!

Osoittimet (ja taulukot)

Osoittimet

- Kertausharjoitus:

```
{
int *a, *b, *c;
a = malloc(sizeof(int));
*a = 10;
b = a;
c = malloc(sizeof(int));
*c = *a;
*a = 23;
}
```

```
{
struct node
{
int data;
struct node *next;
} *s, *t;
s = malloc(sizeof(struct node));
s->data = 12;
t = malloc(sizeof(struct node));
t->next = s;
t->data = 23;
s->next = NULL;
free(t); free(s);
}
```

Osoittimet (ja taulukot)

- Sama asia voidaan sanoa monin eri sanoin:
 - $x[i] = 1;$
 - $*(x + i) = 1;$
 - $*(i + x) = 1;$
 - $i[x] = 1;$
- C:ssä taulukon rajojen ylittämistä ei tarkasteta!

Merkkijonot 1

- C:ssä merkkijonoille ei ole omaa tietotyyppiä (vrt. String Javassa)
- Käytetään merkkijonotaulukoita, jotka päättyvät merkkiin '\0' (null-merkki)
 - käsitteellisesti tämä on eri asia, kuin NULL, vaikkakin molemmat voidaan esittää kokonaisluvulla 0
- Merkkijonovakiot esitetään lainausmerkkien avulla
 - ”merkkijono”

Merkkijonot 2

- Merkkijonotaulukko voidaan alustaa merkkijonovakion avulla
 - `char string[] = "merkkijono";`
 - `'\0'` lisätään automaattisesti
 - `char string[] = {'m','e','r','k','k','i','j','o','n','o','\0'};`

Merkkijonot 2

- Merkkijonotaulukko voidaan alustaa merkkijonovakion avulla
 - `char string[] = "merkkijono";`
 - `'\0'` lisätään automaattisesti
 - `char string[] = {'m','e','r','k','k','i','j','o','n','o','\0'};`
- Merkkijonomuuttujan arvoksi ei voi sijoittaa merkkijonovakiota
 - `string = "uusi";`
 - kyseessä on vain osoitin

Merkkijonot 2

- Merkkijonotaulukko voidaan alustaa merkkijonovakion avulla
 - `char string[] = "merkkijono";`
 - `'\0'` lisätään automaattisesti
 - `char string[] = {'m','e','r','k','k','i','j','o','n','o','\0'};`
- Merkkijonomuuttujan arvoksi ei voi sijoittaa merkkijonovakiota
 - `string = "uusi";`
 - kyseessä on vain osoitin
- Merkkijonoja ei luonnollisesti voi myöskään vertailla `==` operaattorin avulla
 - `Osoitin == viittaustyyppi == Olio tai taulukko Javassa`
 - Eihän olioitakaan verrata `==` operaattorilla

Edelliseltä luennolta

```
void foo(char *a, char *b) {  
    while ( *a++ == *b++ );  
}
```

Merkkijonotaulukot

- `char **stringArray = {"Hei", "Hoi", "Maailma"};`
- `int main(int argc, char** argv) {
 ...
}`
- `./foo`