

T-106.3100, Ohjelmoinnin jatkokurssi T2

Luento 2:
Osoittimista

Petri Ihantola

Tänään luvassa

- Tikulla silmään...
- Direktiivit
- Osoittimet, taulukot ja hieman muistinhallintaa

Edelliseltä luennolta

- Endians

Arvo 0x12345678 tallennettuna eri endian-formaateissa

	big endian	little endian
	12 34 56 78	78 56 34 12
	-----	-----
byte #	1 2 3 4	1 2 3 4

- Bittitason endian harvoin mielekäs käsite
- Muistin varaaminen pinosta ja keosta
 - Tähän palataan myöhemmin tänään
- Tietorakennekirjastot
 - C++ tarjoaa geneerisen tietorakenne kirjaston (STL)
 - C ei tarjoa

Direktiivit

- Alkavat aina #-merkillä
- Ohjeita esikäntäjälle, joka siis tuottaa C-koodia
 - `#include <tiedosto> /* standardikirjastot oletushakemistosta */`
 - `#include "tiedosto" /* nykyisestä hakemistosta ensin */`
 - `#define PI 3.1415926 /* Makrot ISOILLA */`
 - `#undef PI /* purkaa aikaisemman määrittelyn */`
 - `#define M /* arvoa ei ole pakko määritellä */`
 - `#define NIMI(parametrit) ARVO`
 - `#if, #ifdef, #ifndef /* ehdollinen kääntäminen */`
 - `#else /* edellisten pari */`
 - `#endif /* edellisten pari */`

Makroesimerkki

```
#define SQR(x) (x * x)
```

```
...
```

```
int y;
```

```
int x=3;
```

```
y = SQR(x);
```

Makroesimerkki

```
#define SQR(x) (x * x)
```

```
...
```

```
int y;
```

```
int x=3;
```

```
y = SQR(x+1);
```

Makroesimerkki

```
#define SQR(x) (x * x)
```

```
...
```

```
int y;
```

```
int x=3;
```

```
y = SQR(x+1); /* x+1*x+1 */
```

Makroesimerkki

```
#define SQR(x) ((x) * (x))
```

```
...
```

```
int y;
```

```
int x=3;
```

```
y = SQR(x+1); /* ((x+1)*(x+1)) */
```

Makroesimerkki

```
#define SQR(x) ((x) * (x))
```

```
...
```

```
int y;
```

```
int x=3;
```

```
y = SQR(x++);
```

Makroesimerkki

```
#define SQR(x) ((x) * (x))
```

```
...
```

```
int y;
```

```
int x=3;
```

```
y = SQR(x++); /* ((x++)*(x++)) */
```

```
/* Makrojen sivuvaikutukset voivat siis olla ongelmallisia */
```

```
/* myös varatuista sanoista voi tehdä makroja! */
```

Ennalta määritellyjä makroja

- `__LINE__` nykyinen rivinumero
- `__FILE__` tiedoston nimi
- `__TIME__` aika
- `__STDC__` 1, jos ANSI C

Header-tiedoston tyypillinen rakenne

```
#ifndef TIEDOSTO_H_INCLUDE
#define TIEDOSTO_H_INCLUDE
/* header-tiedoston varsinainen sisältö */
...
#endif
```

Ehdollinen kääntäminen

```
#if 0
```

```
/* sisäkkäiset kommentit eivät toimi, mutta tämä  
on näppärä tapa leikata koodia väliaikaisesti pois  
käännöksestä */
```

```
#endif
```

```
#if LINUX /* alustariippuvuuksien huomioiminen */
```

```
#include "linux.h"
```

```
#else
```

```
#include "generic.h"
```

```
#endif
```

Ehdollinen kääntäminen / debuggaus

```
#ifdef DEBUG
```

```
    fprintf(stderr, "size of binary heap= %d\n",h[1]);
```

```
    /* debug-koodi on helppo kytkeä päälle tai pois */
```

```
#endif
```

```
/* debug tulostuksia voi määritellä myös monilla tasoilla */
```

```
#if DEBUG > 1
```

```
    if (...)
```

```
        fprintf (stderr, "Error: " at %s,line %d.\n",
```

```
                __FILE__, __LINE__ );
```

```
#endif
```

Ehdollinen kääntäminen / debuggaus

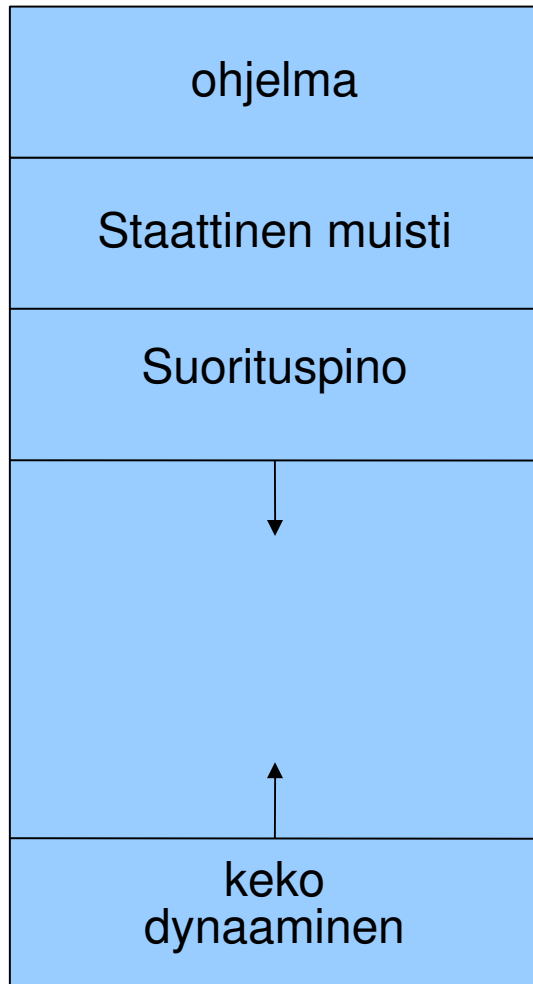
```
include <assert.h>
```

```
assert(ehto)
```

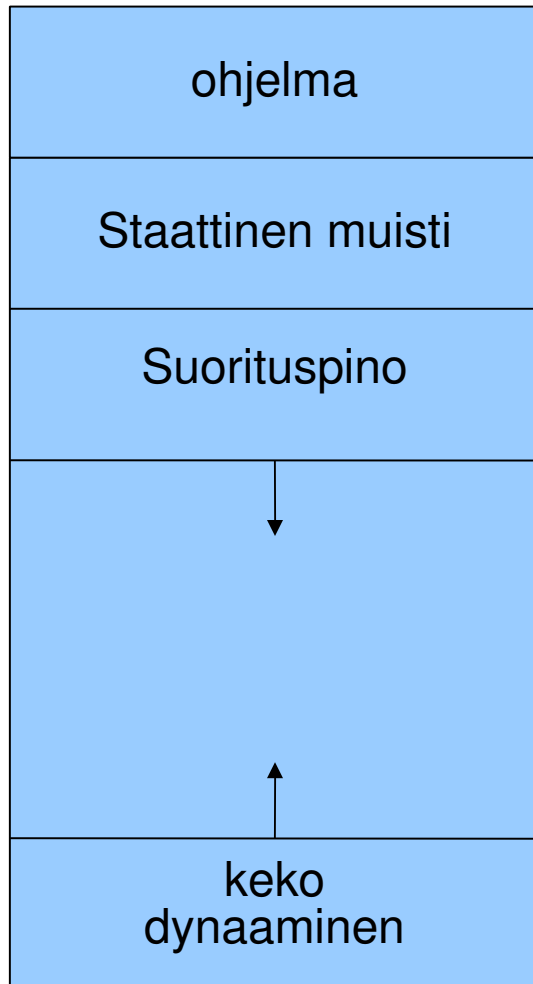
- Mikäli ehto on epätosi (0) tulostaa stderriin virheen, josta selviää tiedoston nimi ja rivinumero ja keskeyttää suorituksen
- Assertiot ohitetaan, mikäli makro NDEBUG on määritelty
 - Makroja voidaan määritellä myös kääntäjän komentoriviargumenttien avulla
 - gcc -DNDEBUG ...

Suorituspino ja pinokehys

Miltä ohjelma näyttää muistissa

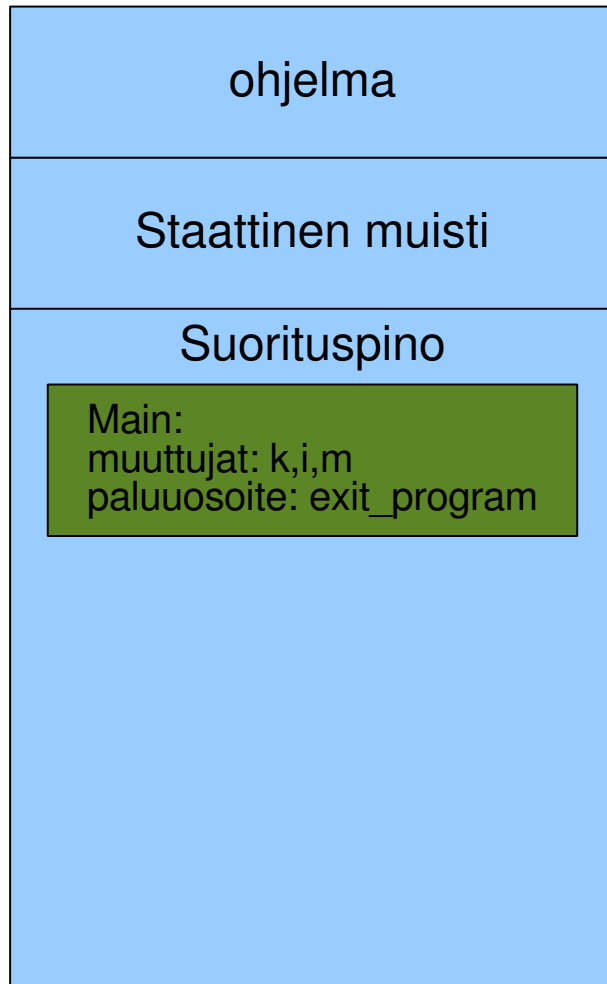


Suorituspino ja pinokehys



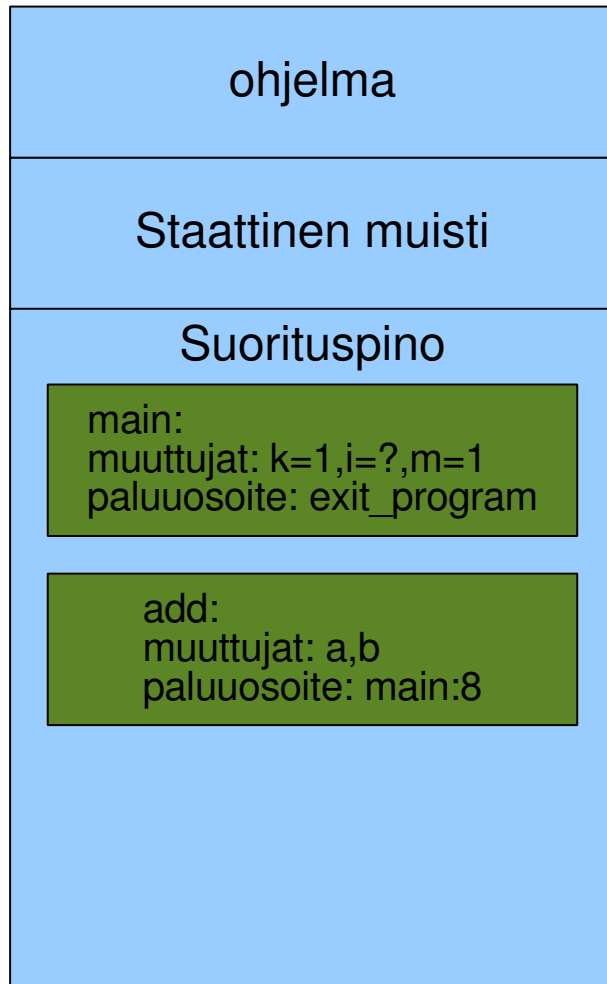
```
#include <stdio.h>
int add(int a,int b);
main() {
    int k = 1;
    int i;
    int m = 1;
    i = add(m,k);
    printf("i = %d\n",i);
}
int add(int a,int b) {
    if(b==0)
        return(a);
    else
        return(1+add(a,b-1));
}
```

Suorituspino ja pinokehys



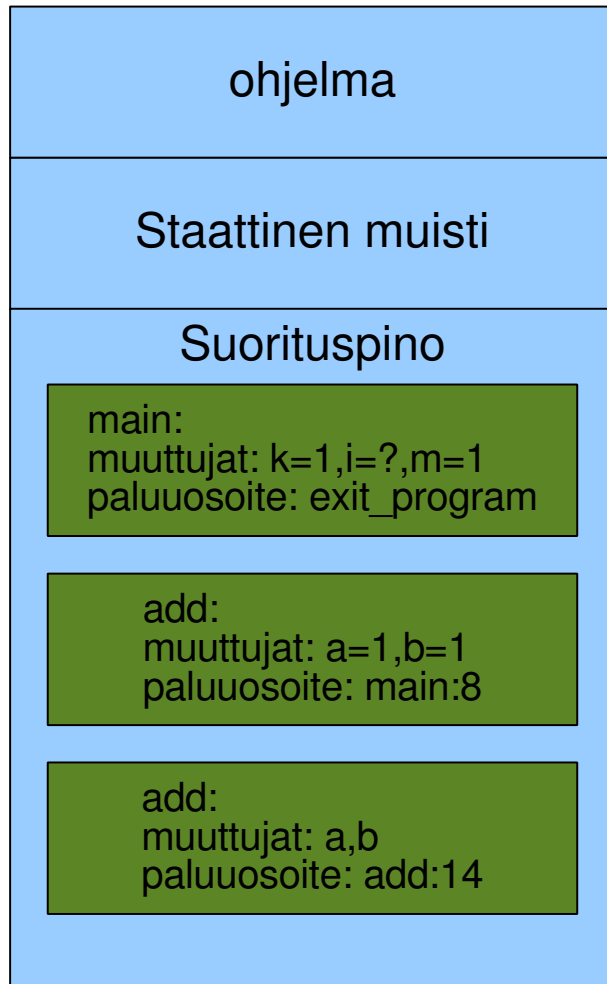
```
#include <stdio.h>
int add(int a,int b);
main() {
    int k = 1;
    int i;
    int m = 1;
    i = add(m,k);
    printf("i = %d\n",i);
}
int add(int a,int b) {
    if(b==0)
        return(a);
    else
        return(1+add(a,b-1));
}
```

Suorituspino ja pinokehys



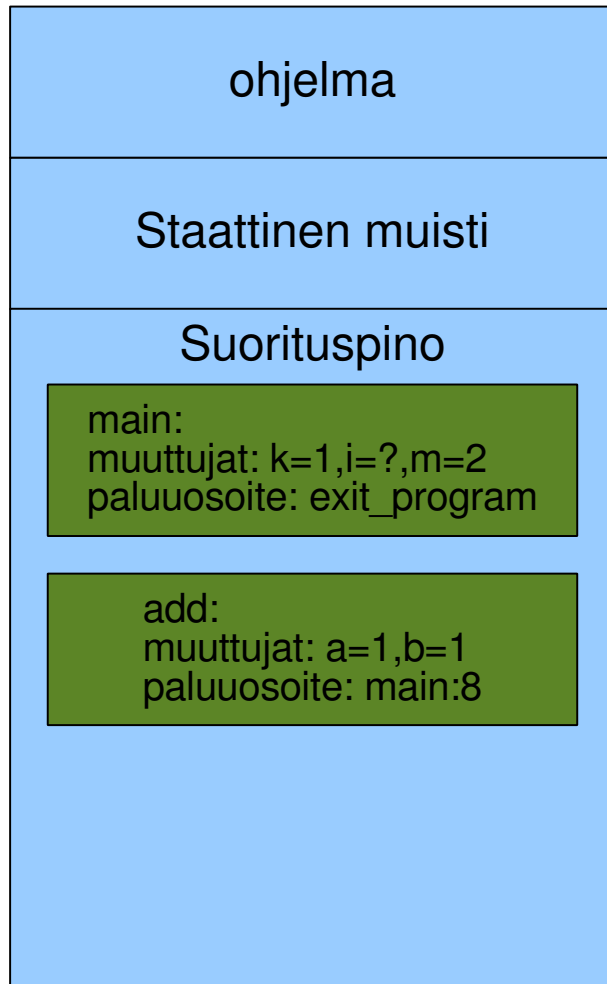
```
1: #include <stdio.h>
2: int add(int a,int b);
3: main() {
4:     int k = 1;
5:     int i;
6:     int m = 1;
7:     i = add(m,k);
8:     printf("i = %d\n",i);
9: }
10:int add(int a,int b) {
11:     if(a==0)
12:         return(b);
13:     else
14:         return(1+add(b,a-1));
15:}
```

Suorituspino ja pinokehys



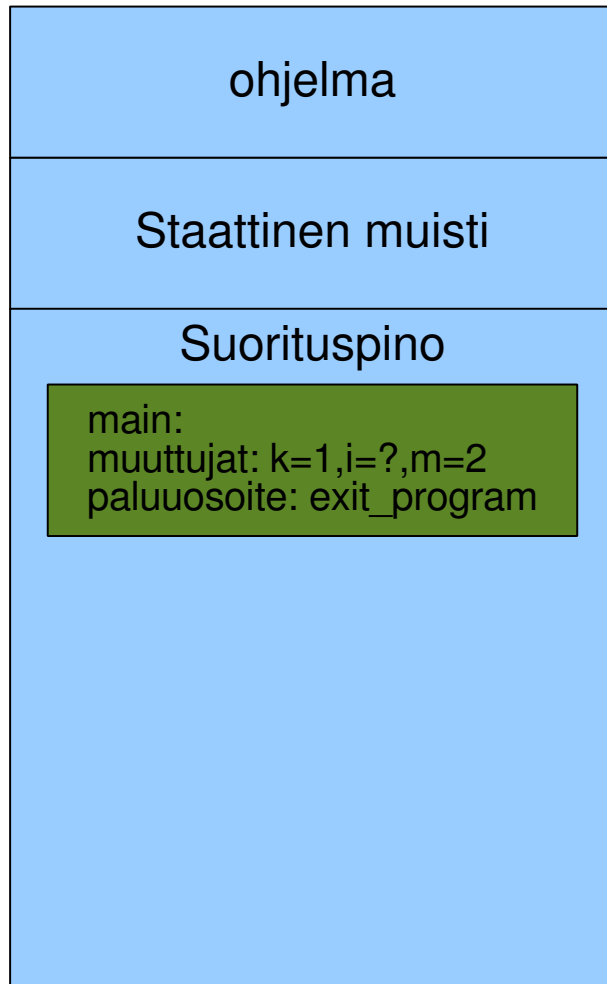
```
1: #include <stdio.h>
2: int add(int a,int b);
3: main() {
4:     int k = 1;
5:     int i;
6:     int m = 1;
7:     i = add(m,k);
8:     printf("i = %d\n",i);
9: }
10:int add(int a,int b) {
11:     if(a==0)
12:         return(b);
13:     else
14:         return(1+add(a,b-1));
15: }
```

Suorituspino ja pinokehys



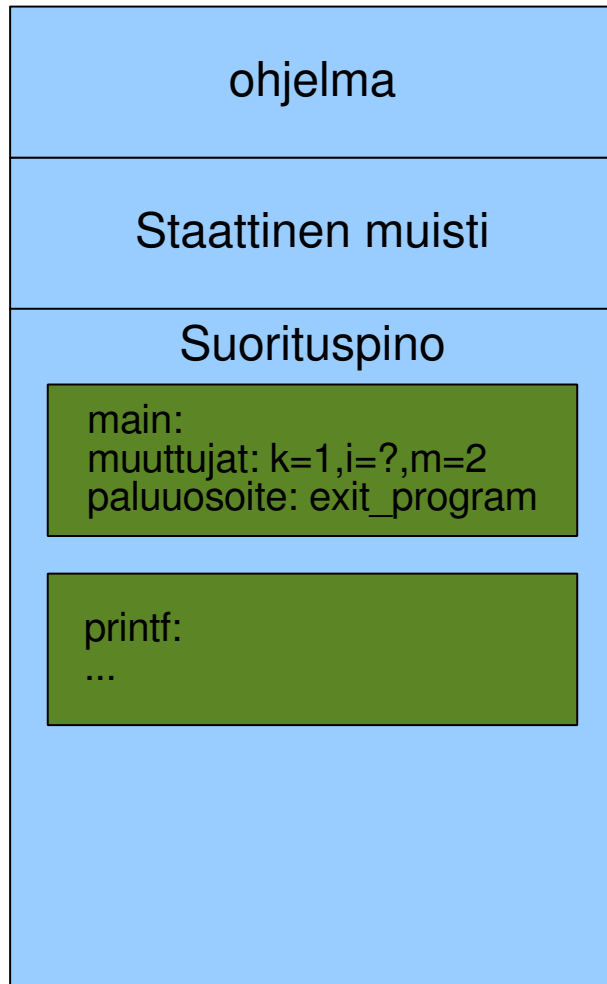
```
1: #include <stdio.h>
2: int add(int a,int b);
3: main() {
4:     int k = 1;
5:     int i;
6:     int m = 1;
7:     i = add(m,k);
8:     printf("i = %d\n",i);
9: }
10: int add(int a,int b) {
11:     if(a==0)
12:         return(b);
13:     else
14:         return(1+add(a,b-1));
15: }
```

Suorituspino ja pinokehys



```
1: #include <stdio.h>
2: int add(int a,int b);
3: main() {
4:     int k = 1;
5:     int i;
6:     int m = 1;
7:     i = add(m,k);
8:     printf("i = %d\n",i);
9: }
10:int add(int a,int b) {
11:     if(a==0)
12:         return(b);
13:     else
14:         return(1+add(a,b-1));
15:}
```

Suorituspino ja pinokehys



```
1: #include <stdio.h>
2: int add(int a,int b);
3: main() {
4:     int k = 1;
5:     int i;
6:     int m = 1;
7:     i = add(m,k);
8:     printf("i = %d\n",i);
9: }
10:int add(int a,int b) {
11:     if(a==0)
12:         return(b);
13:     else
14:         return(1+add(a,b-1));
15: }
```

Suorituspino ja pinokehys

- Pinokehys varataan aina funktiota kutsuttaessa
- Edellinen esimerkki ei ehkä silti vastannut todellisuutta:
 - Häntärekursion poisto
 - Rivinumerot riittämättömät kuvaamaan paluuosoitetta
 - Kokekielinen ohjelma
 - Todellisuudessa pinokehys on monimutkaisempi
 - Paikallisten muuttjien tallentaminen rekistereihin
 - Entäs kun rekisterit eivät riitä

Keko

- Ohjelmoijan eksplisiittisesti varattava muistia ja C:ssä myös vapautettava varattu muisti
- Muistin pirstoutuminen
- Esimerkiksi Javassa oliolle (viittaustyyppisten muuttujien kohteille) varataan tilaa keosta

Osoitin

- Javan viittausmuuttujat ovat C:ssä osoittimia (eng. pointer)
 - Binky Pointer Fun Video: <http://cslibrary.stanford.edu/104/>

Osoitin

- Javan viittausmuuttujat ovat C:ssä osoittimia (eng. pointer)
 - Binky Pointer Fun Video: <http://cslibrary.stanford.edu/104/>
 - Osoitin on muuttuja, jonka arvona on toisen muuttujan osoite
 - `int *a; /* osoitinmuuttuja kokonaislukuun */`
 - `char* a, b; /* vain a on osoitinmuuttuja */`
 - Jokaisesta tyypistä voidaan tehdä osoitin
 - `malloc/calloc/realloc` varaa muistia keosta
 - `sizeof`
 - `free`
 - Esim: `foo() { int* a = malloc(sizeof(int)); ...}`

Osoitinesimerkkejä

- Määrittelyjä osoitettavan tyylin avulla:
 - `char *p; /* osoitin sizeof(char) kokoiseen alueeseen */`
 - `int **q; /* osoitin sizeof(int*) kokoiseen alueeseen */`
- Typedefin avulla
 - `typedef int* IntPtr;`
`IntPtr p,q; /* eri asia, kuin int* p,q */`

Osoittimien käyttöä

```
int i, j; /* kokonaislukumuuttujia */
int *ip; /* osoitinmuuttuja kokonaislukuun */

i = 10;
j = 20;
ip = &i; /* ip osoittaa i:hin */
*ip = 5; /* i=5 */
ip = &j;
*ip += 7;
i += *ip;
```

Osoittimien käyttöä

```
int i, j; /* kokonaislukumuuttujia */
int *ip; /* osoitinmuuttuja kokonaislukuun */

i = 10;
j = 20;
ip = &i; /* ip osoittaa i:hin */
*ip = 5; /* i=5 */
ip = &j; /* ip osoittaa j:hin */
*ip += 7; /* j = 27 */
i += *ip; /* i = 32 */
```

Tämä ja muita hyviä esimerkkejä:

<http://www.comp.lancs.ac.uk/computing/users/ss/java2c/diffs.html>

Pinosta vai keosta

```
int *foo() {  
    int a=1;  
    return &a;  
} /* roikkuva osoitin, dangling pointer */
```

```
int *bar() {  
    int *a;  
    a = malloc(sizeof(int));  
    *a = 1;  
    return a;  
}
```

Taulukot, osoittimet ja osoitinaritmetiikkaa

- Taulukon nimi on samalla sen nollannen alkion osoite
 - Taulukon alkioihin voidaan viitata osoittimilla laskemalla etäisyys ensimmäisestä alkioista

```
int taulu[100];
```

- Taulukon alkioihin voidaan nyt viitata osoittimilla
 - `taulu[10]` ja `*(taulu+10)` tarkoittavat samaa
 - Osoitinaritmetiikka = Käyttäjän ei tarvitse huolehtia siitä, montako tavua `int` on, jotta voisi laskea yllä olevan kaltaisia siirtymiä
 - ```
double *a, *b;
b = a + 5;
```

      - `b` osoittaa jonnekin, joka on  $5 \times$  (doublen vaatima tila) kauempana `a`:sta
      - Mitä siellä sitten on?
- Vaikka taulukon nimi onkin osoitin, sen arvoa ei voi muuttaa

# Taulukot, osoittimet ja osoitinaritmetiikkaa

- Huolimattomalla osoitinaritmetiikalla saa ohjelman ja lukijan helposti sekaisin

# Muistin varaaminen ja vapauttaminen

```
int *ip;
```

```
/* muista varautua muistinvarauksen epäonnistumiseen */
if ((ip = malloc(n*(sizeof(int))) == NULL)
 /* virhe */
```

```
...
```

```
/* muista vapauttaa varaamasi muisti */
free(ip);
ip = NULL; /* "ylimääräistä" siivoamista */
```

# Helppoja virheitä

```
Foo() {
 int taulu[100];
 ...
 free(taulu);
}
```