HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

C-ohjelmointi: Osoittimet

Liisa Marttinen & Tiina Niklander
22.2.2005

Muistinhallinta

Java vs C

- Luokka on viittaustyypin määritelmä ja olio on viittaustyypin ilmentymä
- Muistinhallinta on implisiittistä. Java suoritusympäristö varaa muistia uusille olioille aina oliota luotaessa
- Roskienkeruu vapauttaa muistia eli poistaa muistista ne oliot, joihin ei enää ole viittauksia
- Mikä sitten on olioviite???? (No se on oikeastaan osoitin)

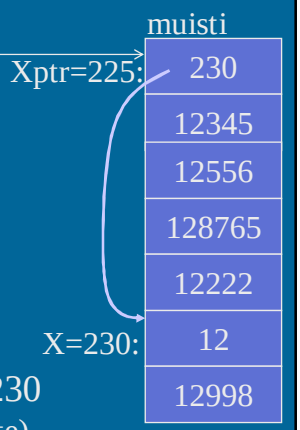
- Ei luokkia, mutta kuitenkin tietorakenteita ja osoittimia
- Eksplisiittinen muistinhallinta. Ohjelman on varattava muistia uusille tietorakenteille.
- Eksplisiittinen vapaus. Ohjelman on vapautettava tarpeeton muisti.
- Osoittimet ovat oikeastaan vain viittauksia muistissa oleviin tietoalkioihin

C osoittimet

- Osoitin on muuttuja, jonka arvona on toisen muuttujan osoite.
- Osoittimet ja taulukot ovat läheistä sukua. Osoittimia voi käyttää kuten taulukoita ja taulukon nimeä voi käyttää kuten vakio-osoitinta (const)
- Osoittimien käyttö perustuu siihen, että useimpien tietokoneiden muisti voidaan kuvata ikään kuin valtavana yksiulotteisena taulukkona. Kaikki ohjelmat ja data sijaitsevat tässä taulukossa.
- Osoitin on oikeastaan indeksi johonkin kohtaan muistia.
- Osoitinmuuttuja on eräänlainen viitta, jonka avulla päästään käsiksi toiseen muuttujaan.

Tieto ja sen osoite (3)

```
Xptr  DC   0
X     DC   12
      LOAD  R1, =X        ; R1 ← 230
      STORE R1, Xptr
      LOAD  R2, X         ; R2 ← 12
      LOAD  R3, @Xptr     ; R3 ← 12
```



- Muuttujan X osoite on 230
- Muuttujan X arvo on 12
- Osoitinmuuttujan (pointterin) Xptr osoite on 225
- Osoitinmuuttujan Xptr arvo on 230
  - jonkun tiedon osoite (nyt X:n osoite)
- Osoitinmuuttujan Xptr osoittaman kokonaisluvun arvo on 12

C-kieli: Y = *ptrX /* ei prtX:n arvo, mutta ptrX:n osoittaman muuttujan arvo */

TIETO

Operaatiot

p = &c	osoitteen otto
c = *p	osoitetun muuttujan arvo
c = **r	-"- (Nyt vain kaksi osoitinta peräkkäin)
p = q	samantyyppiset osoittimet
p+i	p taulukko, i sopivan kokoinen kok.luku
p-i	
p-q	saman taulukon osoittimia ja q<p
p < q	p ja q saman taulukon osoittimia
p == q	
*ip++	osoittimen arvo kasvaa yhdellä
(*ip)++	osoitetun muuttujan arvo kasvaa yhdellä!

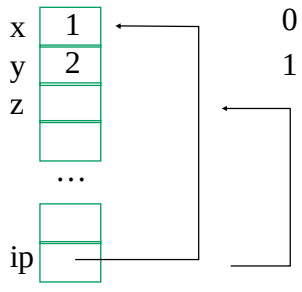
KOODI ESIMERKKI:

```
int main(int argc, char**
argv)
{
 int x=1, y=2, z[10];
 int *ip;
int *p, q;
int *r, *s;

ip = &x;
y = *ip;  /* y = x =1 */
*ip = 0;   /* x = 0 */
ip = &z[0];
}

double atof(char * string);
```

HUOM: p on osoitin ja q tavallinen kokonaisluku



Funktion parametrina osoitin on erittäin tavallinen

Taulukot c:ssä

- Taulukko on vain jono peräkkäisiä olioita, joilla on yhteinen nimi. Esim. int a[15] on 15 peräkkäisen kokonaisluvun jono.
- Taulukon nimi on osoitin, jonka arvona on ensimmäisen alkion osoite. Näin nimi osoittaa ensimmäiseen alkioon.
- Osoitinhan oli muuttuja, jonka arvona on muistiosoite.
- Sijoitus

  pa = &a[0]

  asettaa pa:n arvoksi a:n ensimmäisen alkion osoitteen
- Koska tuo on myös a:n arvo, voidaan kirjoittaa

  pa = a
- Osoitin käyttäytyy kuin taulukko. Syy osoitinaritmetiikka.

Osoitinaritmetiikka

- Osoittimen voidaan katsoa osoittavan yhtenäisen alkiojonon ensimmäiseen alkioon (kuten taulukko)
- Alkioon i viitataan lisäämällä i osoitinmuuttujan arvoon:

  pa+i
- Osoitinaritmetiikka toimii viitattavan tyypin koosta riippumatta. pa+1 viittaa aina seuraavaan samantyyppiseen alkioon.
- Alkion arvo saadaan siis lausekkeella

  *(pa+i)
- Yleensä käytetään taulukkomaista viittausta

  pa[i]
- Osoitimen arvo NULL ei osoita mihinkään

Taulukko parametrina

- Funktiolle välitetään arvoparametrina kopio taulukon nimestä, eli osoite ensimmäiseen alkioon.
- Tästä seuraa, että taulukot ovat muuttujaparametreja ja funktio voi muuttaa alkioiden sisältöjä.
- Funktion parametrina
  char s[]
  char *s
  merkitsevät samaa asiaa.
- Käytä jälkimmäistä epäselvyyksien vuoksi. Koska kyseessä on osoitinmuuttuja eli taulukon nimi.
- HUOM: Parametrissa ei ole taulukon kokoa, koska kyseessä on osoitin eikä muistialue!

Esimerkki 2: merkkijonon kopiointi Käytetään taulukoita!

```
#include <stdio.h>

void kopioi( char *s, char *t)
{
   int i =0;
  while (  (s[i] = t[i]) != '\0' )
    i++;
}

int main (void)
{
  char taalta [] ="Tämä kopioidaan.",
        tanne[50];
  kopioi ( tanne, taalta);    printf("%s\n", tanne);
  kopioi ( tanne, taalta);    printf("%s\n", tanne);
  return 0;
}
```

Merkkijonotaulukot parametrina. Oikeasti osoittimet taulukoiden alkuihin.

Viitataan taulukon alkioihin yksi kerrallaan

Esimerkki 2: merkkijonon kopiointi - jatkuu
Käytetään osoittimia!

Versio 1:

```
void kopioi( char *s, char *t)
{
  while (  (*s = *t) != '\0' )
    s++; t++;
}
```

Versio 2:

```
void kopioi( char *s, char *t)
{
  while (  (*s++ = *t++) != '\0' )
    ;
}
```

Funktion prototyyppi on identtinen taulukkoversion kanssa.

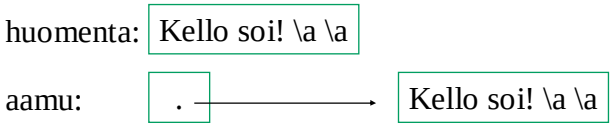
Versio 3:

```
void kopioi( char *s, char *t)
{
  while  ( *s++  =  *t++ )   ;
}
```

Minimalistisen selkeä!

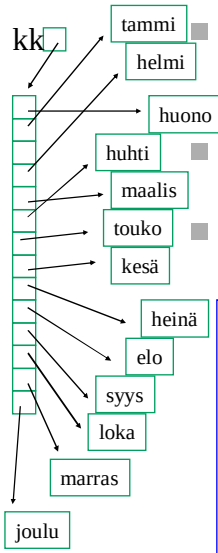
Merkkijonovakiot ja osoittimet

- Merkkijonovakio on merkkitaulukko, jonka päättää lopetusmerkki '\0'.
- char *aamu = "Kello soi! \a \a"
- char huomenta[] = "Kello soi! \a \a"



- Taulukon (huomenta) yksittäisiin alkioihin voidaan viitata
- Osoitin aamu voidaan asettaa muualle, mutta osoitetun merkkijonon muutosyritysten tulos on 'epämääräinen'

Osoitintaulukot



- Koska osoitin on muuttuja, myös sen arvo voidaan osoittaa. Näin saadaan osoitin osoittimeen:
  int **a;
- Samoin voidaan määritellä taulukollinen osoittimia:
  int *t[10];
- Nämä ovat suhteellisen harvinaisia paitsi komentoriviparametrien käsittelyssä.

```
char *kk_nimi(int k)
{
  static char *kk[] = {”huono”, ”tammi”, ”helmi”,
maalis”, ”huhti”, ”touko”, ”kesä”, ”heinä”, ”elo”,
”syys”, ”loka”, ”marras”, ”joulu”};
return ( (k<1 || k > 12)  ? kk[0] : kk[k] );
}
```

Funktio-osoittimet

- Vaikka funktiot eivät ole muuttujia, niin niilläkin on osoite muistissa (kts. esim. TTK-91 konekieli). Tähän osoitteeseen voi viitata funktio-osoittimen avulla.
- Osoitin määritellään seuraavasti
  paluutyyppi (*nimi) (parametrilista);

  int (*lfptr) (char[], int);
  lfptr = getline; /* kun int getline(char s[], int len); */
- Funktio-osoittimen avulla, voidaan eri kutsukerroilla kutsua eri funktiota. Näin voidaan rakentaa yleisiä aliohjelmia. (Esim. stdlib.h:ssa on funktio qsort, joka saa parametrina järjestysfunktion)

Funktiotaulukko: include/linux/quota.h

```
231 /* Operations which must be implemented by each quota format */
232 struct quota_format_ops {
233         int (*check_quota_file)(struct super_block *sb, int type);
             /* Detect whether file is in our format */
234         int (*read_file_info)(struct super_block *sb, int type);
             /* Read main info about file - called on quotaon() */
235        int (*write_file_info)(struct super_block *sb, int type);
             /* Write main info about file */
236        int (*free_file_info)(struct super_block *sb, int type);
              /* Called on quotaoff() */
237         int (*read_dqblk)(struct dquot *dquot);
             /* Read structure for one user */
238         int (*commit_dqblk)(struct dquot *dquot);
             /* Write structure for one user */
239         int (*release_dqblk)(struct dquot *dquot);
             /* Called when last reference to dquot is being dropped */
240 };
```

Tietueet (tai rakenteet)

- Tietue on kokoelma muuttujia (kenttiä), jotka voivat olla keskenään eri tyyppisiä. Rakenteeseen viitataan yhdellä nimellä.

  struct nimi { kentät; };
- Esimerkki: määrittely ei vielä varaa tilaa

  ```
  struct point {
      int x;
      int y;
  }
  ```
- Muuttujat määritellään käyttäen rakenteen nimeä:

  struct point p1, p2, p3;
- Rakenteen kenttään viitataan operaattorilla piste '.'

  p1.x = 2 * p2.y;

Rakenteille sallitut operaatiot

- Voidaan kopioida toisen rakenteen sisällöksi (p1=p2)
- Osoite voidaan ottaa &-operaattorilla
- Kenttiin voidaan viitata .-operaattorilla
- Voidaan alustaa määrittelyvaiheessa
  struct point p4 = {1, 5};

```
typedef struct LogItemType {
        TRID trid;
        int oper;
        int len;
        char data[256];
} LogItemType;
```

```
LogItemType logitems[200];
```

Osoittimet rakenteisiin

- Osoittimet rakenteisiin ovat samanlaisia kuin muutkin osoittimet.
  struct point *pp;
  …
  pp = &p4;
  printf(”orig (%d, %d)\n”, (*pp).x, (*pp).y);
  …
- Osoittimet ovat niin yleisiä, että tuo muoto (*mtuja).kenttä on lyhennetty muotoon muuttuja->kenttä. Yo. uusiksi
  printf(”orig (%d, %d)\n”, pp->x, pp->y);
- Operaattorit ., ->, [] ja () ovat ylimmällä presedenssitasolla ja ryhmitellään vasemmalta oikealle.
- Käytä sulkuja, jos olet epävarma

sizeof ja union

- Varatun tilan saa selville sizeof-operaattorilla (funktiolla)

```
sizeof(struct point);
sizeof(p1);
sizeof(&p1);
sizeof(double);
```

- Union on rakenteen erikoistapaus

```
union u_tag {
  int ival;
  float fval;
  char * sval; };
```

- Arvot jakavat saman muistialueen eli käytännössä ovat vaihtoehtoisia.

malloc, calloc ja free

void * on yleinen osoitin, joka voi osoittaa millaiseen rakenteeseen tahansa.

- void *malloc (size_t size);
  - varaa muistia size tavua ja palauttaa osoittimen varatun muistin alkuun. malloc palauttaa NULL, jos muistin varaus ei onnistu.
- void *calloc (size_t nobj, size_t size);
  - * calloc on kuten malloc, mutta varattavan muistin määrä on nobj*size tavua ja muisti nollataan.
- void *realloc (void *p, size_t size);
  - * realloc muuttaa parametrina annetun osoittimen p osoittaman varatun muistialueen kokoa.
- void free (void *p);
  - * free vapauttaa dynaamisesti varatun muistin takaisin käyttöjärjestelmälle.

Paluuarvon tarkistus!

- Hyvään ohjelmointitapaan kuuluu, että aina tarkistetaan funktion palauttama arvo ja toimitaan sen mukaan.
- Erityisen tärkeää tämä on mm. muistin varauksen yhteydessä.

```
if (k=malloc(sizeof(double)))
    error; /* varaus epäonnistui,  toivuttava tai lopetettava */

/* muistin varaus onnistui ja voidaan jatkaa normaalisti */
```

Abstrakti tietorakenne

- Abstrakti tietorakenne toteuttaa oman toiminnallisuutensa funktioiden ja omien rakenteiden kautta. Oliomaailmassa näitä vastaavat luokat ja niiden metodit, mutta c:ssä ohjelmoija on vastuussa niiden käytöstä.
- Void –tyyppi mahdollistaa geneerisen listan, pinon, hajautustaulun yms. toteutuksen. Todellinen alkio määräytyy tyyppimuunnoksen kautta.
- Toteutetaan yleensä omaan .c - tiedostoonsa (tai kirjastoonsa) ja sillä on usein oma .h –tiedoston käyttäjiään varten
- Esimerkiksi linuxin kernelin koodissa on vaikkapa tiedosto include/linux/list.h (joka tarjoaa listan toteutuksen) (kts. www-sivu http://lxr.linux.no/source/include/linux/list.h)

Osoittimiin liittyviä käytänteitä (osa 1)

- Geneeristä osoitinta (void *p) voidaan tyyppimuunnoksen avulla käyttää käsiteltäessä tietyn tyyppistä muuttujaa
  *(double *)p
- Muistinvaraus n kappaleelle kokonaislukuja
  int *p;
  if ((p=malloc(n*sizeof(int))) == NULL)
    error;
- Muistin vapautus: muista aina tehdä free(p); p=NULL;
- i. alkio
  p[i] (mieluimmin kuin *(p+i))
- Lohkon läpikäynti
  for (pi = p; pi < p+SIZE; pi++)
    käytä läpikäynnissä osoitinta pi

Osoittimiin liittyviä käytänteitä (osa 2)

- Viiteparametri
  1. Prototyypissä muodollisena parametrina osoitin
     void f(int *pp)
  2. Funktion sisällä käytä osoittimen osoittamaa arvoa
     *pp
  3. Kutsussa todellisena parametrina tav. muuttuja
     f(&mtja);
  4. Kutsussa todellisena parametrina osoitinmuuttuja
     f(osmtja);
- Tietuetaulukon läpikäynti
  for (p = block; p < block + n*elSize; p+= elSize)
- i. tietue
  p = block + i*elSize

C assumes that programmer is intelligent enough to use all of its constructs wisely, and so few things are forbidden.

C can be a very useful and elegant tool. People often dismiss C, claiming that it is responsible for a ”bad coding style”. The bad coding style is not the fault of the language, but is controlled (and so caused) by the programmer.