

# C-ohjelmointi

## Luento 6: tietueet ja joukot

21.2.2006  
Tiina Niklander

## Sisältö

- Tietueet (struct)
  - Määrittely
  - Rakenne
  - Käyttö
- Lueteltu tyyppi (enum)
- Joukot (union)
  - Määrittely
  - Rakenne
  - Käyttö
- Linkitetty lista

## Tietueet (struct)

- Tietueessa voi olla vain datakenttiä (vrt. luokka)
  - Tietueen kentät voivat olla keskenään erilaisia.
  - Kenttiä ei voi piilottaa, vaan ne kaikki näkyvät (vrt. *public*).
- Ajatellaan hetkinen Javan luokkatoteutusta Pikkuvarasto (A.Wiklan Java-kurssi). Tällä luokalla on metodi *vieVarastoon*.
  - Metodikutsu *x.vieVarastoon(y)* koskettaa kahta oliota:
  - "This", tässä *x* ja
  - *y*, joka välitettiin parametrina
- C:ssä voi vastaavaa toimintaa jäljitellä, mutta tuo "this" on välitettävä funktiolle parametrina, joten C:ssä vastaava funktio olisi muotoa *vieVarastoon(x,y)*;

## Tietue - Määrittely

```
struct info {  
    char firstName[20];  
    char lastName[20];  
    int age;  
};  
struct info i1, i2;
```

```
typedef struct InfoT {  
    char firstName[20];  
    char lastName[20];  
    int age;  
} InfoT;  
InfoT p1;
```

Suosittelavin tapa

- Tietueen kenttiin viitataan muodolla *nimi.kenttänimi*:
  - *p1.age = 18;*
  - *printf("%s\n", i2.firstName);*

```
struct info {  
    char firstName[20];  
    char lastName[20];  
    int age;  
} k1, k2;
```

## Tietue - Rakenne

- Tietueelle varataan yhtenäinen muistialue.
- Kuitenkin muistialueen koko voi olla eri kuin kenttien yhteenlasketut koot, koska viittausten yksinkertaistamiseksi kenttien välejä saatetaan jättää käyttämättä.
- Esim:  
`sizeof(InfoT) ?>= 40*sizeof(char)+sizeof(int)`

## Tietue - Käyttö

- `InfoT i1, i2;`
- Tietueen sijoittaminen toiselle `i1 = i2`  
kopioi tietueen sisällön bitti kerrallaan täsmälleen samanlaisena, mutta ei seuraa kentissä mahdollisesti olevia osoittimia.
- Tietueita **EI** voi verrata suoraan: `i1 == i2`
- vaan vertailu on tehtävä itse:  

```
strcmp(i1.firstName, i2.firstName) == 0 &&  
strcmp(i1.lastName, i2.lastName) == 0 &&  
i1.age == i2.age
```

## Sisäkkäiset tietueet

- Tietueen kenttinä voi olla myös tietueita

e1:

|         |                      |                      |
|---------|----------------------|----------------------|
| info:   | firstName:           | <input type="text"/> |
|         | lastName:            | <input type="text"/> |
|         | age:                 | <input type="text"/> |
| salary: | <input type="text"/> |                      |

e1.info.age = 21;  
e1.salary = 125.6;

```
typedef struct {
    char firstName[20];
    char lastName[20];
    int age;
} InfoT;
typedef struct {
    InfoT info;
    double salary;
} EmployeeT;
EmployeeT e1;
```

## Osoitin tietueeseen

- Tietueita käsitellään usein osoitinmuuttujan kautta: (\*p).x tai p->x

```
typedef struct pair {
    double x;
    double y;
} PairT, *PairTP;
PairT x;
PairTP p;
```

```
PairT w;
PairTP q;
PairTP p = &w;

if((q = malloc(sizeof(PairT))) == NULL) ...
if((q = malloc(sizeof(struct pair))) == NULL) ...
w.x = 2;
p->x = 1;      (*p).x = 1;      *p.x = 1;
q->y = 3.5;
```

## Tietueet ja funktiot: tietue viiteparametrina

```
void constructorP(PairTP this,
                 double x, double y) {
    this->x = x;
    this->y = y;
}

PairT w;
PairTP p;

constructorP(&w, 1, 2);

if((p = malloc(sizeof(PairT))) == NULL)
    error;
constructorP(p, 1, 2);
```

Funktio saa osoittimen  
Kutsujan aiemmin  
varaaman tietueen  
muistialueeseen.

## Tietueet ja funktiot: tietue paluuarvona

- Funktio voi palauttaa kokonaisen tietueen. Silloin kutsujan on tehtävä palautetusta tietueesta kopio, koska alkuperäinen vapautuu pinosta funktiosta palattua.

```
PairT constructorFunc(double x, double y) {
    PairT p;
    p.x = x;
    p.y = y;
    return p;
}

PairT w = constructorFunc(1, 2.2); /* kopio */
```

Pinossa palautettu  
tietue on kopioitava heti  
talteen.

## Tietueet ja funktiot: osoitin paluuarvona

- Funktio varaa tilan tietueelle, jolloin kutsujan vastuulle jää vapauttaa tuo varattu tila.

```
PairTP constructor(double x, double y) {
    /* client responsible for deallocation */
    PairTP p;
    if((p = malloc(sizeof(PairT))) == NULL)
        return NULL;
    p->x = x;
    p->y = y;
    return p;
}

PairTP p1 = constructor(1, 2);
free(p1);
```

Funktio varaa tilan tietueelle  
Ja palauttaa osoittimen siihen  
Kutsuja vapauttaa tilan  
myöhemmin.

## Tietueet ja funktiot: osoitin paluuarvona

- Käyttö toisen funktion parametrina:

```
int compare(const PairTP p, const PairTP q)
{
    return p->x == q->x && p->y == q->y;
}

PairTP p2 = constructor(1, 3);
PairTP p3 = constructor(2, 6);
int i = compare(p3, p2);
free(p2); free(p3);
```

- Vältä muistivuotoa!

```
i = compare(p1, constructor(3.5, 7));
```

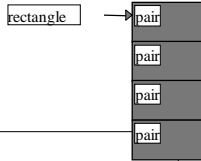
Tässä kadotetaan  
osoitin tietueeseen,  
joten muisti jää  
vapauttamatta

## Muistilohko tietueista

- Muistilohkon käsittely ei riipu lohkon alkioiden tyyppistä.
- Tietueen omiin alkioihin viittaus kuten itsenäisissäkin tietueissa.

```
PairTP rectangle;
PairTP aux;
double x, y;

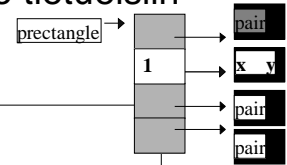
if((rectangle= malloc(4*sizeof(PairT))!=NULL)error;
for(aux = rectangle; aux < rectangle + 4; aux++) {
printf("Enter two double values:");
if(scanf("%lf%lf", &x, &y) != 2) /* error */
break;
constructorP(aux, x, y);
}
```



## Osoitinlohko tietueisiin

- Mahdolliset viittaustavat:
  - prectangle[1][0].x
  - prectangle[1]->x
  - (prectangle+1)->x

```
int i;
PairTP *prectangle;
for(i = 0; i < 4; i++) {
printf("Enter two double values:");
if(scanf("%lf%lf", &x, &y) != 2)
error;
if((prectangle[i] = constructor(x, y))
== NULL)
error;
}
for(i = 0; i < 4; i++)
printf("vertex %d = (%f %f)\n", i,
prectangle[i][0].x, prectangle[i][0].y);
```



## Lueteltu tyyppi (enum)

- Luetellun tyyppin määrittelemät järjestetyt vakiot vastaavat järjestyksessä kokonaislukuja 0,1,2 jne.
- Numeroinnin voi myös aloittaa haluamasta arvosta

```
typedef enum opcodes {
lvalue, rvalue,
push, plus
} OpcodesT;

enum opcodes e;
OpcodesT f;

int i = (int)rvalue; /*i=1*/
```

```
enum opcodes {
lvalue = 1, rvalue,
push, plus
};

enum opcodes e = lvalue;
if(e == push) ...

int i = (int)rvalue; /*i=2*/
```

## Lueteltu tyyppi funktion paluuarvona

- Lueteltua tyyppiä voi käyttää virhetiedon käsittelyssä
- Virheilmoitustekstit kootaan taulukkoon, jota indeksoidaan luetellun tyyppin alkioita vastaavilla kokonaislukuarvoilla

```
typedef enum {
FOPEN, FCLOSE, FOK
} FoperT;

#define TOINT(f) ((int)(f))

char *Messages[] = {
"File can not be opened",
"File can not be closed",
"Successful operation",
"This can not happen"
};
```

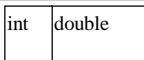
```
FoperT process();

printf("result of calling process() is %s\n",
Messages[TOINT(process())]);
```

## Joukot (union)

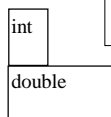
- Tietue
  - Alkiot peräkkäin eli kaikki käytettävissä
- Joukko
  - Alkiot päällekkäin eli vaihtoehtoisia

```
struct intAndDouble {
int i;
double d;
};
```



intAndDouble

```
union intOrDouble {
int i;
double d;
};
```



intOrDouble

## Joukko - Käyttö

- Käyttö yleensä tietueen osana
- Tietueessa kenttä (tag), joka kertoo miten joukko pitää tulkita
- Käytetään paljon tietoliikenneprotokollissa säästämässä tilaa
- Kenttiin viitataan samalla pistenotaatiolla kuin tietueidenkin kenttiin

```
typedef enum {
integer, real
} TagTypeT;

typedef struct {
TagTypeT tag;
union {
int i;
double d;
} value;
} TaggedValueT;
TaggedValueT v;

if(v.tag == integer)
...v.value.i...;
else
...v.value.d...;
```

## Tauko

## Linkitetty tietorakenteet

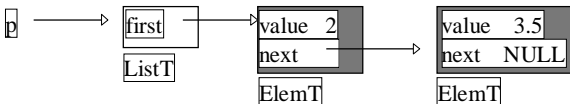
- Abstraktit tietorakenteet
  - Käsitelty Tietorakenteiden kursilla
  - Kannattaa kerrata kaikki rakenteet sieltä
- Pino : operaatiot push, pop ja empty
- Jono : operaatiot enqueue, dequeue ja empty
- Linkitetty lista – rakenteesta päätettävä
  - Yhteen suuntaan vai kahteen suuntaan
  - Rengas vai ei
  - Tunnussolmu vai ei
  - Järjestetty vai järjestämätön
  - Alkiot erilaisia vai sallitaan myös samanlaiset alkiot

## Linkitetty lista - Määrittely

- Tunnussolmullinen yhteensuuntaan linkitetty
- NULL arvoa käytetään aina listan lopun osoittamiseen
- Huomaa seuraavaan alkioon osoittavan kentän next määrittely!

```
typedef double DataType;
typedef struct elem {
    DataType value;
    struct elem *next;
} ElemT, *ElemTP;

typedef struct {
    ElemTP first;
} ListT, *ListTP;
ListTP p;
```



## Linkitetty lista – Tunnussolmun luonti ja poisto

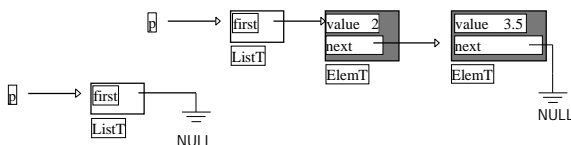
- Listan luonti
  - Tehdään vain tunnussolmu ja asetetaan linkit
- Listan poistaminen
  - Alkioiden poistoon oma funktio
  - Tässä vain tunnussolmun tilan vapautus

```
ListTP construct(void) {
    ListTP p;
    if((p = malloc(sizeof(ListT)))
        == NULL)
        return NULL;
    p->first = NULL;
    return p;
}
```

```
void destruct(ListTP *this) {
    clear(*this); /* alkiot pois */
    free(*this);
    *this = NULL;
}
```

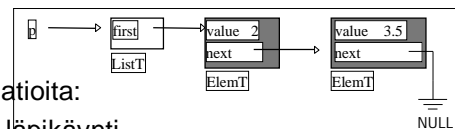
## Linkitetty lista - Käyttö

- Kaikkien listaa käsittelevien toimintojen täytyy säilyttää seuraavat *invariantit*:
  - Tyhjälle listalle pätee  $p \rightarrow \text{first}$  on NULL.
  - Epätyhjälle listalle pätee, että viimeisen alkion next-kentän arvo on NULL.



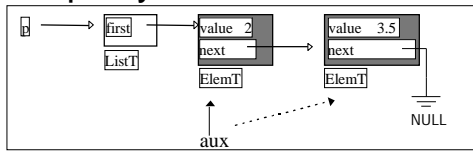
## Linkitetty lista - Käyttö

- Operaatioita:
- Listan läpikäynti
  - Aina linkien kulkusuuntaan
- Lista lisääminen
  - Alkuun, loppuun, keskelle
- Listasta poistaminen
  - Alusta, lopusta, keskeltä



## Listan läpikäynti

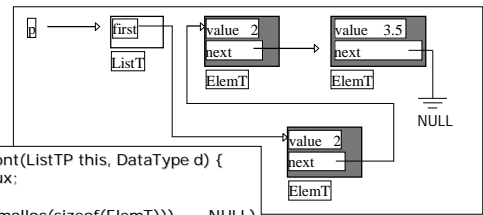
idioms



```
void printAll(const ListTP this) {
    ElemTP aux;
    for(aux=this->first; aux!=NULL; aux=aux->next)
        printf("%f\n", aux->value);
}
```

```
/* pidetään aux edellisessä alkiossa */
If ((p->first != NULL) && (p->first->next !=NULL))
for(aux = p->first; aux->next != NULL;
    aux = aux->next)...
```

## Lisäys listan alkuun

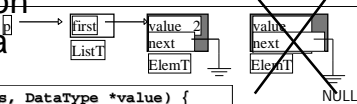


```
int insertFront(ListTP this, DataType d) {
    ElemTP aux;

    if((aux = malloc(sizeof(ElemT))) == NULL)
        return 0;

    aux->next = this->first; /* save state */
    aux->value = d;
    this->first = aux;
    return 1;
}
```

## Viimeisen alkion poisto listasta



```
int deleteLast(ListTP this, DataType *value) {
    ElemTP aux;

    if(this->first == NULL) /* empty list */
        return 0;
    if(this->first->next == NULL) { /* single */
        *value = this->first->value;
        free(this->first);
        this->first = NULL;
        return 1;
    }
    for(aux = this->first; aux->next->next != NULL;
        aux = aux->next)
        ; /* aux viimeistä edelliseen */
    *value = aux->next->value;
    free(aux->next);
    aux->next = NULL;
    return 1;
}
```

## Koko listan tuhoaminen

- Tuhotaan alkio kerrallaan
- edetään lista alusta loppuun

```
int deleteFirst(ListTP this) {
    ElemTP aux = this->first;
    if(aux == NULL) /* empty list */
        return 0;
    this->first = aux->next;
    free(aux);
    return 1;
}

void clear(ListTP this) {
    while(deleteFirst(this))
        ;
    this->first = NULL;
}
```

## Moduuli List

- Tehdään oma käännoyksikkö (moduuli), joka sisältää listan määrittelyt ja käsittelyfunktiot
- Tässä tehtävällä moduulilla on seuraavat piirteet
  - Moduuli pystyy käsittelemään useita eri listoja
  - Moduuli käsittelee void-tyyppistä dataa, joten eri listoihin voi sijoittaa eri tyyppistä tietoa
  - Listan sisäinen toteutus ei näy käyttäjälle
  - Listan käyttäjä vastaa datan rakenteesta ja käsittelystä
- Tyyppien määrittelyt:
  - ListItem ja List
- Funktiot:
  - CreateList, CreateItem, AddTail, AddHead, ListLength, DeleteList, PrintList, EmptyList

Lähde: Jan Lindströmin verkkomoniste

## Moduulin tarjoama rajapinta: list.h

```
#ifndef MY_LIST_LIBRARY
#define MY_LIST_LIBRARY
/* Määritellään listatyyppit */
typedef struct listitem {
    struct listitem *Next; /* Seuraava alkio listassa */
    struct listitem *Prev; /* Edellinen alkio listassa */
    void *Data; /* Tietoalkio */
    unsigned long Size; /* Tietoalkion koko */
} ListItem;
typedef struct {
    ListItem *Head; /* Listan alku */
    ListItem *Tail; /* Listan loppu */
    unsigned long Items; /* Listan alkioden lkm */
} List;
```

## List.h jatkuu

```
/* Listakirjaston tukemat funktiot */
extern List *CreateList(void); /* Luo uusi lista */
extern ListItem *CreateItem(void *Data,
    unsigned long Size); /* Luo lista-alkio */
extern int AddTail(List *,ListItem *);
    /* Lisää listan loppuun */
extern int AddHead(List *,ListItem *);
    /* Lisää listan alkuun */
extern unsigned long ListLength(List *);
    /* Laske listan pituus */
extern void DeleteList(List *); /* Tuhoa lista */
extern void PrintList(List *);
    /* Tulosta listan sisältö */
extern int EmptyList(List *);
    /* Tarkista onko lista tyhjä */
#endif
```

## List.c : CreateList

```
/*
Varataan muistia uudelle listalle ja alustetaan kentät
*/
List *CreateList(void)
{
    List *uusi;

    if(!(uusi = (List *)malloc(sizeof(List))))
        return NULL;

    uusi->Head = NULL;
    uusi->Tail = NULL;
    uusi->Items = 0;

    return uusi;
}
```

## List.c: CreateItem

```
/* Varataan muistia uudelle listan alkioille
ja alustetaan kentät */
ListItem *CreateItem(void *Data,unsigned long size)
{
    ListItem *uusi;
    /* Jos järkevää dataa ei ole annettu poistu */
    if (Data == NULL)
        return NULL;
    if(!(uusi = (ListItem *)malloc(sizeof(ListItem))))
        return NULL;
    if(!(uusi->Data = (void *)malloc(size)))
        return NULL;
    uusi->Next = NULL;
    uusi->Prev = NULL;
    memcpy(uusi->Data,Data,size);
    uusi->Size = size;
    return uusi;
}
```

## List.c: AddTail

```
/* Lisätään alkio listan loppuun */
extern int AddTail(List *lista,ListItem *item)
{
    if (lista == NULL || item == NULL )
        return 1;
    if ( lista->Head == NULL)
        lista->Head = lista->Tail = item;
    else
    {
        lista->Tail->Next = item;
        item->Prev = lista->Tail;
        lista->Tail = item;
    }
    lista->Items++;
    return 0;
}
```

## List.c: AddHead

```
/* Lisätään alkio listan alkuun */
extern int AddHead(List *lista,ListItem *item)
{
    if (lista == NULL || item == NULL )
        return 1;
    if ( lista->Head == NULL)
        lista->Head = lista->Tail = item;
    else
    {
        lista->Head->Prev = item;
        item->Next = lista->Head;
        lista->Head = item;
    }
    lista->Items++;
    return 0;
}
```

## List.c: ListLength ja EmptyList

```
/*
'Lasketaan' listan pituus
*/
unsigned long ListLength(List *lista)
{
    if ( lista == NULL )
        return 0;
    else
        return lista->Items;
}
```

```
/*
Tarkista onko lista tyhjä
*/
int EmptyList(List *lista)
{
    if ( lista == NULL)
        return 1;
    else if (lista->Head == NULL )
        return 1;
    else
        return 0;
}
```

## List.c:DeleteList

```
/* Tuhotaan koko lista */
void DeleteList(List *lista)
{
    ListItem *tmp;
    if ( lista == NULL )
        return;

    tmp = lista->Head;
    while(tmp != NULL)
    {
        tmp = lista->Head->Next;
        free(lista->Head->Data);
        free(lista->Head);
        lista->Head = tmp;
    }
    free(lista);
}
```

## List.c: PrintList

```
/* Tulostetaan koko lista */
void PrintList(List *lista)
{
    ListItem *tmp = lista->Head;

    printf("|");

    while(tmp != NULL)
    {
        printf("%s|", (char *)tmp->Data);
        tmp = tmp->Next;
    }
    printf("|-\n");
}
```

Tässä oletetaan, että listassa on merkkijonoja. Parempi ratkaisu olisi käyttää funktioparametria data-alkioiden käsittelyyn. Käyttäjä joutuisi silloin itse ohjelmoimaan alkioiden käsittelyn, mutta listan käyttöalue kasvaisi.

## Harjoitustyö

- Käytettävät piirteet:
  - Linkitetty tietorakenne osoittimilla (pino, lista, puu, hajautustaulu, ...)
  - Tiedosto (tekstitiedosto tai binääritiedosto)
  - Komentoriviparametrit (jos ei muuta järkevää, niin ainakin -h opastus)
  - Funktioita parametreineen mielekkäästi
- Käännyttävä laitoksen Linux-ympäristössä gcc:n parametreilla -ansi -pedantic ja -Wall ilman varoituksia
- Vähintään kaksi käännösyksikköä ja make
- Koodi dokumentoitava järjevästi
- Erillinen lyhyt rakennedokumentti ja käyttöohje

## Aiheet

1. Yksinkertainen laskin
2. Sanalaskuri
3. Kaupan kassan simulointi
4. Lennon varausjärjestelmä
5. Työntekijärekisteri
6. Ravintolan simulointi
7. Sukupuu
8. Kokkaavan ystävän apu

## Aiheen valinta

- Aihe olisi syytä valita mahdollisimman pian **pe 17.3. mennessä**
- Valinnan voi kertoa sähköpostilla: [Liisa.Marttinen@cs.helsinki.fi](mailto:Liisa.Marttinen@cs.helsinki.fi)
- Tai 2. periodin ensimmäisellä viikolla luennolla tai laskareissa.
- Omista (varsinkin simulointityyppisistä) aiheista voi neuvotella Liisan tai Tiinan kanssa