

## F09 - Egendefinierade datastrukturer och sökning

Programmeringsteknik med C och Matlab, 7,5 hp

Niclas Börlin  
[niclas.borlin@cs.umu.se](mailto:niclas.borlin@cs.umu.se)

Datavetenskap, Umeå universitet

2023-10-12 Tor

## Egendefinierade datastrukturer

### Döpa om typer

- ▶ Det är möjligt att **döpa om** typer i C
- ▶ Det görs med nyckelordet **typedef**

```
typedef int score;
```

- ▶ Sen kan vi använda den i ett program

```
1  #include <stdio.h>
2
3  typedef int score;
4
5  int main(void)
6  {
7      score s;
8
9      // do stuff with s
10
11     return 0;
12 }
```

### Enkelt ritprogram

- ▶ Antag att vi vill skriva ett enkelt ritprogram som hanterar **objekt** som ska ritas ut i **2 dimensioner**
  - ▶ Linjer dras mellan **två punkter**
  - ▶ En punkt består av en **x-koordinat** och en **y-koordinat**
- ▶ **int**, **char**, **double** samt fält låter oss åstadkomma en hel del, men ibland behöver vi mer
- ▶ Sammansatta, **egendefinierade** datatyper låter oss representera samlingar av **flera** objekt av ev. **olika typ** som **ett** objekt

## Egendefinierade poster — struct (1)

### ► Kodan

```
typedef struct {  
    type1 id1;  
    type2 id2;  
    ...  
} type_name;
```

definierar en datatyp med namn `type_name` som är en **post** (eng. *record*, **struct**)

- Posten `type_name` är sammansatt av **delar** som kallas **fält** (eng. *field*, *member*)
  - `id1` av typ `type1`,
  - `id2` av typ `type2`, osv.
- Alla poster av typen `type_name` har **samma** delar
- OBS! Blanda inte ihop fält i poster med **datatypen** fält (*array*)

## Egendefinierade poster — struct (2)

- Efter att vi definierat typen kan vi använda den som om den var inbyggd, deklarerar variabler, skicka som parametrar, osv.

```
#include <stdio.h>  
  
typedef struct {  
    type1 id1;  
    type2 id2;  
    ...  
} type_name;  
  
type_name merge(type_name a, type_name b)  
{  
    ...  
}  
  
int main(void)  
{  
    type_name i, j, k;  
    ...  
    k = merge(i, j);  
    ...  
    return 0;  
}
```

## Posten två-dimensionell punkt

- Till vårt ritprogram behöver vi en datatyp för **punkter** i två dimensioner:

```
typedef struct {  
    double x;  
    double y;  
} point;
```

## Initiering av poster

- **Hela** poster kan **initieras** i samband med att de **deklaras**, men inte senare
- Följande funkar alltså:
- ... men inte

```
#include <stdio.h>  
#include <stdlib.h>  
  
typedef struct {  
    double x;  
    double y;  
} point;  
  
int main(void)  
{  
    point p1 = {2.8, 1.0};  
    ...  
    return 0;  
}
```

```
#include <stdio.h>  
#include <stdlib.h>  
  
typedef struct {  
    double x;  
    double y;  
} point;  
  
int main(void)  
{  
    point p1;  
    p1 = {2.8, 1.0};  
    ...  
    return 0;  
}
```

## Tilldelning, kopiering av post

- En **struct** kan **tilldelas** en annan **struct** av **samma typ**

```
point p1 = {2.8, 1.0}, p2;  
p2 = p1;
```

- En **struct** kan skickas som parameter **till** en funktion och returneras **från** en funktion

## Fältåtkomst

- För att komma åt enskilda fält i en post används **punktoperatoren** (**.**)
- Fältvariabeln fungerar som vilken **variabel som helst** av den typen

```
point p1, p2;  
p1.x = 2.8;  
p1.y = 1.0;  
p2.x = p1.x + 1.0;  
p2.y = p1.y;  
printf("x = %f\n", p1.x);
```

## Konstruktion

- Följande funktion tar två flyttal med koordinater och returnerar en point med de koordinaterna

```
point create_point(double x, double y)  
{  
    point p;  
    p.x = x;  
    p.y = y;  
    return p;  
}
```

## Skriva ut poster

- En **struct** kan inte skrivas ut direkt med **printf**
- I stället får man skriva ut varje enskilt element

```
void print_point(point p)  
{  
    printf("x = %.2f, y = %.2f\n", p.x, p.y);  
}
```

## Läsa in i poster

- ▶ En **struct** kan inte heller läsas in direkt med `scanf`

```
point get_point(void)
{
    point p;
    printf("Give the x coordinate: ");
    scanf("%lf", &p.x);
    printf("Give the y coordinate: ");
    scanf("%lf", &p.y);
    return p;
}
```

## Linjer

- ▶ Vi kan använda vår punktdatatyp till att representera **linjer**

```
typedef struct {
    int id;
    point p1;
    point p2;
} line;
```

- ▶ En post av typen `line` representerar en linje
  - ▶ vars identitetsnummer heter `id`
  - ▶ vars första punkt heter `p1`
  - ▶ vars andra punkt heter `p2`

## Skapa en linje

```
line create_line(int id, point p1, point p2)
{
    line l = {id, p1, p2};
    return l;
}
```

## Skriva ut en linje

```
void print_line(line l)
{
    printf("Line id: = %d\n", l.id);
    printf("Point 1: ");
    print_point(l.p1);
    printf("Point 2: ");
    print_point(l.p2);
}
```

## Fält av poster

- En post kan användas i ett fält, precis som en enkel datatyp

```
int main(void)
{
    point p1 = {2.8, 1};
    point p2 = {3.7, 3.1};
    point p3 = {5.6, 8.2};

    line l1 = {1, p1, p2};
    line l2 = {2, p2, p3};

    const int n = 2;
    line lines[n];
    lines[0] = l1;
    lines[1] = l2;

    print_lines(n, lines);

    return 0;
}
```

## Skriva ut fält av poster

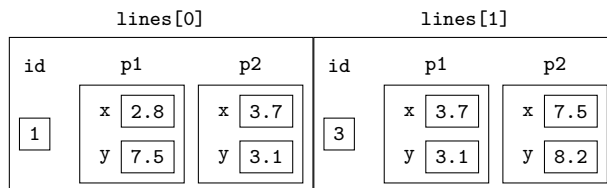
- En fält av poster kan skrivas ut som "vanliga" fält förutom att utskrift av enskild post måste hanteras:

```
void print_lines(int n, line lines[])
{
    for(int i = 0 ; i < n ; i++) {
        print_line(lines[i]);
    }
}
```

## Enskilda fält i fält av poster

- Exempel på att komma åt enskilda fält i ett fält av poster:

```
lines[1].id = 3;
lines[1].p1 = lines[0].p2;
lines[1].p2.x = 7.5;
lines[1].p2.y = 8.2;
```



## Sökning

- ▶ Att kunna söka efter data är viktigt i många tillämpningar
- ▶ Det finns flera olika algoritmer för sökning
- ▶ Vi kommer att titta på två algoritmer
  - ▶ Linjärsökning
  - ▶ Binärsökning

## Linjärsökning

## Linjärsökning

- ▶ Algoritm med vanliga ord:
  - ▶ Starta från början och sök tills elementet hittats eller sekvensen tar slut
- ▶ Algoritm lite mer formellt

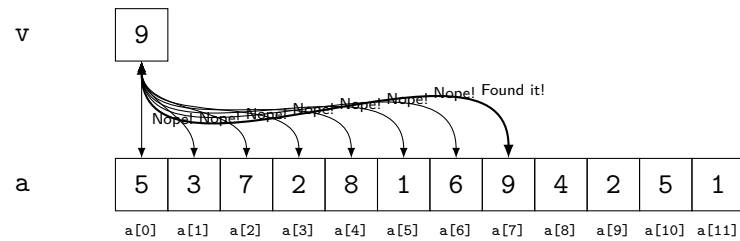
```
1 Algorithm linsearch(a: Array, n: Int, v: Value)
2   i <- 0
3   while i < n do
4     if v = a[i] then
5       return i
6     i <- i + 1
7   return -1
```

- ▶ Notera att algoritmen är lite mer generell än kod i C
  - ▶ Vi har t.ex. inte låst oss vid vilken typ som **Value** motsvarar

## Linjärsökning i C

```
1 int linsearch(const int *a, int n, int v)
2 {
3     int i = 0;
4     while (i < n) {
5         if (v == a[i]) {
6             return i;
7         }
8         i++;
9     }
10    return -1;
11 }
```

## Linjärsökning, exempel



## Binärsökning

## Binärsökning (1)

- Kräver att sekvensen är **sorterad**
- Algoritm med vanliga ord
  1. Jämför med elementet **närmast mitten** i sekvensen
    - 1.1 Om likhet — **klart**
    - 1.2 Om det sökta värdet kommer före elementet närmast mitten, **sök i den vänstra delsekvensen**, hoppa till steg 1
    - 1.3 Om det sökta värdet kommer efter elementet närmast mitten, **sök i den högra delsekvensen**, hoppa till steg 1

## Binärsökning (2)

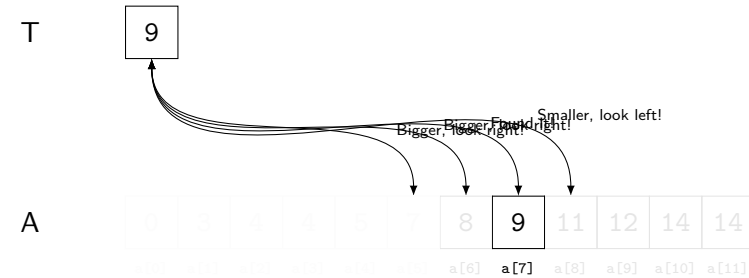
### ► Algoritm lite mer formellt

```
1  Algorithm binsearch(arr: Array, n: Int, v: Value)
2  left <- 0
3  right <- n - 1
4  while left <= right do
5      mid <- (left + right) / 2 // Integer division
6      if v = arr[mid] then
7          return mid // Found it
8      else if v < arr[mid] then
9          right <- mid - 1 // Look left
10     else
11         left <- mid + 1 // Look right
12
13     return -1 // Not found
```

## Binärsökning i C

```
1 int linsearch(const int *a, int n, int val)
2 {
3     int left = 0;
4     int right = n - 1;
5     while (left <= right) {
6         int mid = left + (right - left) / 2;
7         if (val == a[mid]) {
8             // Found it; return index
9             return mid;
10        } else if (val < a[mid]) {
11            // Look left
12            right = mid - 1;
13        } else {
14            // Look right
15            left = mid + 1;
16        }
17    }
18    // Signal not found
19    return -1;
20 }
```

## Binär sökning, exempel



## Sökning

- ▶ Varför olika algoritmer?
  - ▶ Olika effektivitet
  - ▶ Olika tillfällen då de fungerar
- ▶ Det finns fler algoritmer än dessa
- ▶ Mer i kursen *Datastrukturer och algoritmer*

## Annat

- ▶ Kommentarer
  - ▶ Skriv kommentaren helst på raden FÖRE koden
  - ▶ Undvik kommentarer i slutet på en rad, och då bara korta
  - ▶ Inga långa rader! (<120 tecken)
- ▶ Tips: Slösa med utrymme!
  - ▶ Om det finns en maxbegränsning på 10 domare, skapa utrymme för 10 resultat!
  - ▶ Metoder att skapa exakt rätt mängd utrymme kommer att gås igenom på senare kurser!