

Screencast

Arrayer

Sammanfattning

- En array i C är en samling av N element av typen T i konsekutivt minne
- Arrayer i C har en fix storlek som inte kan ändras efter att de skapades
- Syntaxen för att skapa en array "fido" av N element av typ T :

```
 $T$  fido[ $N$ ];
```

```
int foo[128]; // 128 heltal  
char bar[256]; // 256 tecken
```

- Före C99 var N tvunget att vara en konstant, men efter C99 kan N vara en variabel

```
void foo(int n) {  
     $T$  fido[n];  
}
```

- Man kan också allokera ett minnesblock av godtycklig storlek och behandla det som en array

```
malloc(sizeof( $T$ ) * 77)
```

Ovanstående uttryck returnerar en pekare till en array av 77 T -element.

Argumentet till malloc är antalet bytes som skall allokeras, så för att få en array at 77 heltal måste man alltså räkna ut storleken i bytes för ett heltal med hjälp av **sizeof**.

- För att komma åt det N :te elementet i en array x skriver man $x[N]$
- Följande kod loopar över samtliga element i en array och sätter dess värden till 8:

```
int bobby[1024 * 1024];  
for (int index = 0; index < 1024 * 1024; ++index) {  
    bobby[index] = 8;  
}
```

- Följande kod flyttar alla värden i en array "ett steg åt vänster" och ersätter det sista elementet med det första:

```
int length = 1024;  
T values[length];  
T first = values[0];  
for (int index = 0; index < length-1; ++index) {  
    values[index] = values[index+1];  
}  
values[length-1] = first;
```

Notera att arrayer indexeras från 0 (alltså det första elementet har index 0).

- Pekare (introducerades som hastigast i screencast 6) och arrayer är i stort sett utbytbara i C.
- om p är en pekare till ett antal element av typen T i minnet kan man komma åt det 5:e elementet på dessa sätt:

```
p[4]
*(p+4)
```

Det sista är ett exempel på *pekararitmetik* och betyder med start i adressen p , gå $4 \times \text{sizeof}(T)$ bytes ”till höger”, och läs T -värdet på denna minnesplats

Följande modifikation av föregående program är alltså laglig:

```
int length = 1024;
T *values = malloc(sizeof(T) * 1024);
T first = values[0];
for (int index = 0; index < length-1; ++index) {
    values[index] = values[index+1];
}
values[length-1] = first;
```

- Likaså denna (programmet är logiskt ekvivalent med det förra):

```
int length = 1024;
T *values = malloc(sizeof(T) * length);
T first = *values;
for (T *cursor = values; cursor < values+length; ++cursor) {
    *cursor = *(cursor+1);
}
*(values+length-1) = first;
```

- Notationen `p[4]` är *syntaktiskt socker* för `*(p+4)`, och ovanstående kodexempel torde illustrera varför man behöver en enklare syntax!
- Data i C-program innehåller inga metadata, så en array har ingen aning om sin längd
- Programmeraren måste själv hålla reda på längden i programmet (se exempelprogrammen ovan)
- Ett program som indexerar sig utanför en array har ett odefinierat beteende
- Flerdimensionella arrayer är möjliga, t.ex. `int matrix[10][10]` för en 10×10 -matris som accessas med `matrix[5][7] = -1`, etc.