UNIVERSITETET I OSLO
DET MATEMATISK-NATURVITENSKAPELIGE FAKULTET

Dagens tema

- Vektorer (array-er)
- Tekster (string-er)
- Adresser og pekere
- Dynamisk allokering

INF1070

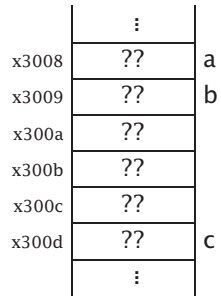
Vektorer

Alle programmeringsspråk har mulighet til å definere en såkalte **vektor** (også kalt **matrise** eller «array» på engelsk). Dette er en samling variable av samme type hvor man bruker en **indeks** til å skille dem.

Deklarasjon

I C deklareres vektorer ved å sette antallet elementer i hakeparenteser etter variabelnavnet:

```
char a, b[4], c;
```



Antallet elementer må være en *konstant*.

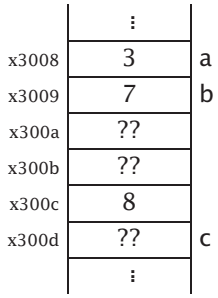
INF1070

Bruk

Ved bruk angir indeksen hvilket element vi ønsker. I C er alltid første element nr. 0, neste nr. 1, osv.

```
a = 3;
b[0] = 7;  b[a] = 8;
```

Etter dette er situasjonen:



INF1070

Beregning av adresse

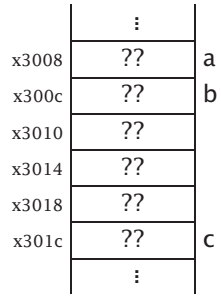
Adressen til vanlige variable er kjent[†] men adressen til vektorelementer må beregnes. Formelen er

$$\mathit{Startadresse} + \mathit{Indeks} \times \mathit{Størrelse}$$

Størrelse over 1 byte

Anta at int er 4 byte.

```
int a, b[4], c;
```



Hva skjer med ulovlig indeks?

I C sjekkes ikke indeksen. Dette gjør det mulig å ødelegge andre variable, kode eller i noen tilfelle hele systemet.

[†] Dette er ikke helt sant, men vi kan tro det er slik en stund.

INF1070

Tekster

I C lagres tekster som tegnvektorer med en spesiell konvensjon: Etter siste tegn står en byte med verdien 0.†

Variable

Når man deklarerer en tekstvariabel, må man angi hvor mange tegn det er plass til (samt plass til 0-byten).

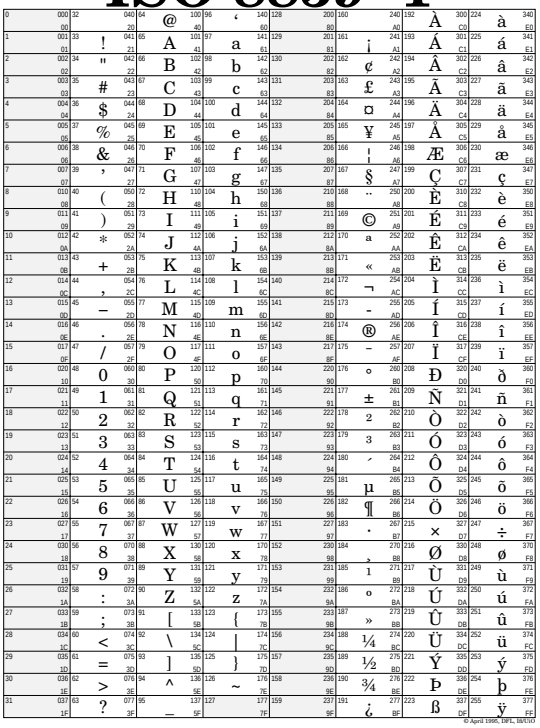
```
char str[6];
```

Tekstvariabel str har plass til 5 tegn.

† En byte med verdien 0 er ikke det samme som sifferet «0»; det er representert av verdien 48, som vist på neste lysark.

INF1070

ISO 8859–1



Kopiering av tekst

Flytting av tekst skjer med standardfunksjonen strcpy:

```
strcpy(str, "abc");
```

	Før			Etter	
	⋮			⋮	
x3010	??	str	x3010	'a'	str
x3011	??		x3011	'b'	
x3012	??		x3012	'c'	
x3013	??		x3013	0	
x3014	??		x3014	??	
x3015	??		x3015	??	
x3016	'a'	"abc"	x3016	'a'	"abc"
x3017	'b'		x3017	'b'	
x3018	'c'		x3018	'c'	
x3019	0		x3019	0	
	⋮			⋮	

INF1070

Andre tekstoperasjoner

strlen(str) beregner den nåværende lengden av teksten i str. (Dette gjør den ved å lete seg frem til 0-byten.)

strcat(str1,str2) utvider teksten i str1 med den i str2.

strcmp(str1,str2) sammenligner de to tekstene. Returverdien er

< 0 om str1 < str2

0 om str1 = str2

> 0 om str1 > str2

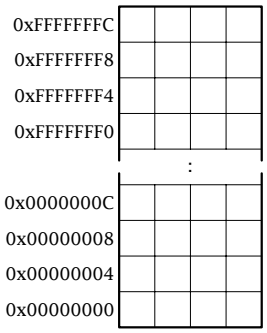
sprintf(str, "...", v1, v2, ...) fungerer som printf men resultatet legges i str i stedet for å skrives ut.

Hva om teksten er for lang?
Siden tekstvariable er vektorer, er det ingen sjekk på plassen. Det er derfor fullt mulig å ødelegge for seg selv (og noen ganger for andre).

INF1070

Variable, adresser og pekere

Variable ligger lagret i *hurtiglageret* (ofte kalt *RAM*) i en eller annen adresse.



INF1070

Operatoren &

I C kan man få vite i hvilken adresse en variabel ligger ved å bruke operatoren &.

```
#include <stdio.h>

int a, b, c;

int main(void)
{
    printf("Skriv to tall: ");
    scanf("%d", &a); scanf("%d", &b);
    c = a + b;
    printf("Summen er %d.\n", c);

    printf("I adresse %08x ligger a med verdien %d.\n", &a, a);
    printf("I adresse %08x ligger b med verdien %d.\n", &b, b);
    printf("I adresse %08x ligger c med verdien %d.\n", &c, c);
}
```

INF1070

La oss kjøre dette programmet:

```
Skriv to tall: 47 9
Summen er 56.
I adresse 00020e00 ligger a med verdien 47.
I adresse 00020e04 ligger b med verdien 9.
I adresse 00020e08 ligger c med verdien 56.
```

NB! Det kan variere fra gang til gang hvilke adresser man får.

Her ser vi at variablene ligger pent etter hverandre og at hver av dem opptar 4 byte.

INF1070

Pekervariable

I C kan vi legge adresser i variable; disse deklareres med en stjerne:

```
int v, *p;
```

Her er v en vanlig variabel mens p er en peker som kan peke på int-variable. (Vi må alltid oppgi hva slags variable pekere skal peke på.)

Bruk av pekervariable

Vi kan sette adressen til variable inn i pekervariabelen; vi sier at vi får pekeren til å «peke på» variabelen.

```
p = &v;
```

INF1070

Vi kan «følge en peker» ved å bruke operatoren *; da får vi variabelen som pekeren peker på.

```
v = 7;
printf("v = %d, *p = %d.\n", v, *p);
v = -17;
printf("v = %d, *p = %d.\n", v, *p);
```

Denne koden skriver ut

```
v = 7, *p = 7.
v = -17, *p = -17.
```

Både v og *p angir altså samme variabel:

```
*p = 123;
printf("v = %d, *p = %d.\n", v, *p);
```

Utskriften av denne koden er

```
v = 123, *p = 123.
```

INF1070

Et eksempel
La oss lage en funksjon som bytter om de to parametrene sine.

Til selve ombyttingen trengs en hjelpevariabel:

```
tempVal = firstVal;
firstVal = secondVal;
secondVal = tempVal;
```

INF1070


```
#include <stdio.h>

void Swap(int firstVal, int secondVal);

main()
{
  int valueA = 3;
  int valueB = 4;

  printf("Before Swap: valueA = %d and valueB = %d\n", valueA, valueB);
  Swap(valueA, valueB);
  printf("After Swap : valueA = %d and valueB = %d\n", valueA, valueB);
}

void Swap(int firstVal, int secondVal)
{
  int tempVal;          /* Needed to hold firstVal when swapping */

  tempVal = firstVal;
  firstVal = secondVal;
  secondVal = tempVal;
}
```

INF1070

Når vi kjører programmet, får vi en overraskelse:

```
Before Swap: valueA = 3 and valueB = 4
After Swap : valueA = 3 and valueB = 4
```

Grunnen er: Parametre overføres som verdier i C (som i Java).

Systemet tar altså en kopi av parameterverdien og legger denne i et register (eller et annet sted for parametre).

Følgelig er det bare lokale kopier som endres. Når funksjonen er ferdig, er alt glemt.

INF1070

Løsning
Løsningen er å overføre *pekere* til de to variablene i stedet for verdiene.

Pekerne overføres som kopier, men vi kan allikevel endre det de peker på.

INF1070


```
#include <stdio.h>
void NewSwap(int *firstVal, int *secondVal);
main()
{
    int valueA = 3;
    int valueB = 4;
    printf("Before NewSwap: valueA = %d and valueB = %d\n", valueA, valueB);
    NewSwap(&valueA, &valueB);
    printf("After NewSwap : valueA = %d and valueB = %d\n", valueA, valueB);
}
void NewSwap(int *firstVal, int *secondVal)
{
    int tempVal;        /* Needed to hold firstVal when swapping */
    tempVal = *firstVal;
    *firstVal = *secondVal;
    *secondVal = tempVal;
}
```

INF1070

Legg merke til at både funksjonsdeklarasjonen og kallet er endret!

Når dette programmet kjører, skjer alt som vi forventer:

```
Before NewSwap: valueA = 3 and valueB = 4
After NewSwap : valueA = 4 and valueB = 3
```

Konklusjon om parametre

- Det er ulike måter å overføre parametre på.
- I C og i Java brukes *verdioverføring*.
- Man kan allikevel oppdatere variable ved å sende over *pekere* til dem. Dette gjøres for eksempel i

  scanf("%d", &v);

INF1070

Dynamisk allokering

Ofte trenger man å opprette objekter under kjøringen i tillegg til variablene. Standardfunksjonen malloc («memory allocate») benyttes til dette. Parameter er antall byte den skal opprette; operatoren sizeof kan gi oss dette.

Vi må ha med stdlib.h for at malloc skal fungere skikkelig.

```
#include <stdlib.h>
    ...
int *p;
    ...
p = malloc(sizeof(int));
```

Frigivelse av objekter

Når objekter ikke trengs mer, må de gis tilbake til systemet med funksjonen free:

```
free(p);
```

INF1070

Hva hvis noe går galt?

Følgende Java-program inneholder en feil:

```
1  class Feil {
2      void m() {
3      }
4
5      public static void main (String args[]) {
6          Feil fp = null;
7
8          fp.m();
9      }
10 }
```

Når vi kjører det, får vi beskjed om hva som gikk galt:

```
> javac Feil.java
> java Feil
Exception in thread "main" java.lang.NullPointerException
        at Feil.main(Feil.java:8)
```


Et eksempel
Anta at vi skal lese et navn (dvs en tekst) og skrive det ut. For at navnet ikke skal oppta plass når vi ikke trenger det, bruker vi dynamisk allokering.

```
char *navn;
    :
    printf("Hva heter du? ");
    navn = malloc(200);
    scanf("%s", navn);
    printf("Hei, %s.\n", navn);
    free(navn);
```


Her er et C-program med tilsvarende feil:

```
1  #include <stdio.h>
2
3  int main (void)
4  {
5      char *s;
6
7      strcpy(s, "Abc");
8      return 0;
9  }
```

Når vi kompilerer og kjører det, skjer følgende:

```
> gcc feil.c -o feil
> ./feil
Segmentation fault
```

Konklusjon Vær nøye med å få programmet riktig.

(Vi kommer ellers tilbake med verktøy for feilfinning siden.)