



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 deklarerer vektorer ved å sette antallet elementer i hakeparenteser etter variabelnavnet:

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

	:	
x3008	??	a
x3009	??	b
x300a	??	
x300b	??	
x300c	??	
x300d	??	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:

	:	
x3008	3	a
x3009	7	b
x300a	??	
x300b	??	
x300c	8	
x300d	??	c
	:	

INF1070

Beregning av adresse

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

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

Størrelse over 1 byte

Anta at int er 4 byte.

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

	:	
x3008	??	a
x300c	??	b
x3010	??	
x3014	??	
x3018	??	
x301c	??	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»; sifferet «0» er representert av verdien 48, som vist på neste lysark.

INF1070

ISO 8859-1

000	32	040	04	@	100	96	c	140	128	200	160	240	192	À	300	224	à	240	192
01	00	20	20	À	101	97	a	141	129	201	161	241	193	Á	301	225	á	241	193
02	01	21	21	Â	102	98	b	142	130	202	162	242	194	Â	302	226	â	242	194
03	02	22	22	Ã	103	99	c	143	131	203	163	243	195	Ã	303	227	ã	243	195
04	03	23	23	Ä	104	100	d	144	132	204	164	244	196	Ä	304	228	ä	244	196
05	04	24	24	Å	105	101	e	145	133	205	165	245	197	Å	305	229	å	245	197
06	05	25	25	Æ	106	102	f	146	134	206	166	246	198	Æ	306	230	æ	246	198
07	06	26	26	Ç	107	103	g	147	135	207	167	247	199	Ç	307	231	ç	247	199
08	07	27	27	È	108	104	h	148	136	208	168	248	200	È	308	232	è	248	200
09	08	28	28	É	109	105	i	149	137	209	169	249	201	É	309	233	é	249	201
10	09	29	29	Ê	110	106	j	150	138	210	170	250	202	Ê	310	234	ê	250	202
11	10	30	30	Ë	111	107	k	151	139	211	171	251	203	Ë	311	235	ë	251	203
12	11	31	31	Ì	112	108	l	152	140	212	172	252	204	Ì	312	236	ì	252	204
13	12	32	32	Í	113	109	m	153	141	213	173	253	205	Í	313	237	í	253	205
14	13	33	33	Î	114	110	n	154	142	214	174	254	206	Î	314	238	î	254	206
15	14	34	34	Ï	115	111	o	155	143	215	175	255	207	Ï	315	239	ï	255	207
16	15	35	35	Ð	116	112	p	156	144	216	176	256	208	Ð	316	240	ð	256	208
17	16	36	36	Ñ	117	113	q	157	145	217	177	257	209	Ñ	317	241	ñ	257	209
18	17	37	37	Ò	118	114	r	158	146	218	178	258	210	Ò	318	242	ò	258	210
19	18	38	38	Ó	119	115	s	159	147	219	179	259	211	Ó	319	243	ó	259	211
20	19	39	39	Ô	120	116	t	160	148	220	180	260	212	Ô	320	244	ô	260	212
21	20	40	40	Õ	121	117	u	161	149	221	181	261	213	Õ	321	245	õ	261	213
22	21	41	41	Ö	122	118	v	162	150	222	182	262	214	Ö	322	246	ö	262	214
23	22	42	42	×	123	119	w	163	151	223	183	263	215	×	323	247	×	263	215
24	23	43	43	Ü	124	120	x	164	152	224	184	264	216	Ü	324	248	ü	264	216
25	24	44	44	Ý	125	121	y	165	153	225	185	265	217	Ý	325	249	ý	265	217
26	25	45	45	ÿ	126	122	z	166	154	226	186	266	218	ÿ	326	250	ÿ	266	218
27	26	46	46		127	123		167	155	227	187	267	219		327	251		267	219
28	27	47	47		128	124		168	156	228	188	268	220		328	252		268	220
29	28	48	48	¼	129	125		169	157	229	189	269	221	¼	329	253		269	221
30	29	49	49	½	130	126		170	158	230	190	270	222	½	330	254		270	222
31	30	50	50	¾	131	127		171	159	231	191	271	223	¾	331	255		271	223
32	31	51	51		132	128		172	160	232	192	272	224		332	256		272	224

Kopiering av tekst

Flytting av tekst skjer med standardfunksjonen strcpy:

```
#include <stdio.h>

char *strcpy(char til[], char fra[])
{
    int i = 0;
    while (1) {
        til[i] = fra[i];
        if (fra[i] == 0) return til;
        ++i;
    }
}

int main(void)
{
    char t[10];
    int i;

    strcpy(t, "abc");
    for (i = 0; i < 10; ++i) {
        printf("t[%2d] = %4d = '%c'\n", i, t[i], t[i]);
    }
    return 0;
}
```

INF1070

Før

Etter

x3010	??	t	x3010	'a'	str
x3011	??		x3011	'b'	
x3012	??		x3012	'c'	
x3013	??		x3013	0	
x3014	??		x3014	??	
x3015	??		x3015	??	
x3016	??		x3016	??	
x3017	??		x3017	??	
x3018	??		x3018	??	
x3019	??		x3019	??	
x301a	'a'	"abc"	x301a	'a'	"abc"
x301b	'b'		x301b	'b'	
x301c	'c'		x301c	'c'	
x301d	0		x301d	0	
	⋮			⋮	

INF1070

Her er utskriften fra kjøringen:

```
t[ 0] = 97 = 'a'  
t[ 1] = 98 = 'b'  
t[ 2] = 99 = 'c'  
t[ 3] = 0 = ''  
t[ 4] = -40 = 'Ø'  
t[ 5] = 92 = '\\'  
t[ 6] = 112 = 'p'  
t[ 7] = 0 = ''  
t[ 8] = 120 = 'x'  
t[ 9] = -124 = ''
```

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.

0xFFFFFFFFC				
0xFFFFFFFF8				
0xFFFFFFFF4				
0xFFFFFFFF0				
	:			
0x0000000C				
0x00000008				
0x00000004				
0x00000000				

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("adresse %08x ligger a med verdien %d.\n", &a, a);  
    printf("adresse %08x ligger b med verdien %d.\n", &b, b);  
    printf("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 deklarerer 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:

```
temp = v1;
v1 = v2;
v2 = temp;
```

Her er hele programmet:

```
#include <stdio.h>

void swap (int v1, int v2)
{
    int temp;

    temp = v1;
    v1 = v2;
    v2 = temp;
}

int main (void)
{
    int a = 3;
    int b = 4;

    printf("Før: a = %d og b = %d\n", a, b);
    swap(a, b);
    printf("Etter: a = %d og b = %d\n", a, b);
}
```

INF1070

Når vi kjører programmet, får vi en overraskelse:

```
Før: a = 3 og b = 4
Etter: a = 3 og b = 4
```

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

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

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

Legg merke til at både funksjonsdeklarasjonen og kallet er endret!

```
#include <stdio.h>
void swap (int *v1, int *v2)
{
    int temp;
    temp = *v1;
    *v1 = *v2;
    *v2 = temp;
}
int main (void)
{
    int a = 3, b = 4;
    printf("Før: a = %d og b = %d\n", a, b);
    swap(&a, &b);
    printf("Etter: a = %d og b = %d\n", a, b);
}
```

INF1070

Når dette programmet kjører, skjer alt som vi forventer:

```
Før: a = 3 og b = 4
Etter: a = 4 og b = 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 m0() {
3     }
4     public static void main (String args[]) {
5         Feil fp = null;
6         fp.m0();
7     }
8 }
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)
```

INF1070

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);
```

INF1070

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.)

INF1070