



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»; det er representert av verdien 48, som vist på neste lysark.

INF1070

ISO 8859-1

0	000	32	040	04	@	100	98	c	140	108	100	101	240	192	À	300	224	à	340
1	001	33	041	05	!	101	99	A	141	109	101	102	241	193	Á	301	225	á	341
2	002	34	042	06	''	102	98	B	142	100	102	102	242	194	Â	302	226	â	342
3	003	35	043	07	#	103	99	C	143	101	103	103	243	195	Ã	303	227	ã	343
4	004	36	044	08	\$	104	100	D	144	102	104	104	244	196	Ä	304	228	ä	344
5	005	37	045	09	%	105	101	E	145	103	105	105	245	197	Å	305	229	å	345
6	006	38	046	10	&	106	102	F	146	104	106	106	246	198	Æ	306	230	æ	346
7	007	39	047	11	'	107	103	G	147	105	107	107	247	199	Ç	307	231	ç	347
8	008	40	048	12	(108	104	H	148	106	108	108	248	200	È	308	232	è	348
9	009	41	049	13)	109	105	I	149	107	109	109	249	201	É	309	233	é	349
10	010	42	050	14	«	110	106	J	150	108	110	110	250	202	Ê	310	234	ê	350
11	011	43	051	15	»	111	107	K	151	109	111	111	251	203	Ë	311	235	ë	351
12	012	44	052	16	+	112	108	L	152	110	112	112	252	204	Ë	312	236	ë	352
13	013	45	053	17	,	113	109	M	153	111	113	113	253	205	Ì	313	237	ì	353
14	014	46	054	18	-	114	110	N	154	112	114	114	254	206	Í	314	238	í	354
15	015	47	055	19	.	115	111	O	155	113	115	115	255	207	Î	315	239	î	355
16	016	48	056	20	/	116	112	P	156	114	116	116	256	208	Ï	316	240	ï	356
17	017	49	057	21	0	117	113	Q	157	115	117	117	257	209	Ð	317	241	ð	357
18	018	50	058	22	1	118	114	R	158	116	118	118	258	210	Ñ	318	242	ñ	358
19	019	51	059	23	2	119	115	S	159	117	119	119	259	211	Ò	319	243	ò	359
20	020	52	060	24	3	120	116	T	160	118	120	120	260	212	Ó	320	244	ó	360
21	021	53	061	25	4	121	117	U	161	119	121	121	261	213	Ô	321	245	ô	361
22	022	54	062	26	5	122	118	V	162	120	122	122	262	214	Õ	322	246	õ	362
23	023	55	063	27	6	123	119	W	163	121	123	123	263	215	Ö	323	247	ö	363
24	024	56	064	28	7	124	120	X	164	122	124	124	264	216	×	324	248	×	364
25	025	57	065	29	8	125	121	Y	165	123	125	125	265	217	Ø	325	249	ø	365
26	026	58	066	30	9	126	122	Z	166	124	126	126	266	218	Ù	326	250	ù	366
27	027	59	067	31	:	127	123	[167	125	127	127	267	219	Ú	327	251	ú	367
28	028	60	068	32	;	128	124	\	168	126	128	128	268	220	Û	328	252	û	368
29	029	61	069	33	<	129	125]	169	127	129	129	269	221	Ü	329	253	ü	369
30	030	62	070	34	=	130	126	^	170	128	130	130	270	222	Ý	330	254	ý	370
31	031	63	071	35	>	131	127	~	171	129	131	131	271	223	Þ	331	255	þ	371
1	032	64	072	36	?	132	128	—	172	130	132	132	272	224	ß	332	256	ÿ	372

Kopiering av tekst

Flytting av tekst skjer med standardfunksjonen strcpy:

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

Før		Etter	
x3010	??	x3010	'a'
x3011	??	x3011	'b'
x3012	??	x3012	'c'
x3013	??	x3013	0
x3014	??	x3014	??
x3015	??	x3015	??
x3016	'a'	x3016	'a'
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.

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("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 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:

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

INF1070

```
#include <stdio.h>
void Swap(int firstVal, int secondVal);
int 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;
    tempVal = firstVal;
    firstVal = secondVal;
    secondVal = tempVal;
    /* Needed to hold firstVal when swapping */
}
```

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 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