



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 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*.

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
	⋮	

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.

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.

ISO 8859-1

0	000	32	040	64	@	100	96	‘	140	128	200	160	240	192	À	300	224	à	340
	00		20			40			60		80		A0		À	C0		à	E0
1	001	33	041	65	A	101	97	a	141	129	201	161	241	193	Á	301	225	á	341
	01		21			41			61		81		A1		Á	C1		á	E1
2	002	34	042	66	B	102	98	b	142	130	202	162	242	194	Â	302	226	â	342
	02		22			42			62		82		A2		Â	C2		â	E2
3	003	35	043	67	C	103	99	c	143	131	203	163	243	195	Ã	303	227	ã	343
	03		23			43			63		83		A3		Ã	C3		ã	E3
4	004	36	044	68	D	104	100	d	144	132	204	164	244	196	Ä	304	228	ä	344
	04		24			44			64		84		A4		Ä	C4		ä	E4
5	005	37	045	69	E	105	101	e	145	133	205	165	245	197	Å	305	229	å	345
	05		25			45			65		85		A5		Å	C5		å	E5
6	006	38	046	70	F	106	102	f	146	134	206	166	246	198	Æ	306	230	æ	346
	06		26			46			66		86		A6		Æ	C6		æ	E6
7	007	39	047	71	G	107	103	g	147	135	207	167	247	199	Ç	307	231	ç	347
	07		27			47			67		87		A7		Ç	C7		ç	E7
8	010	40	050	72	H	110	104	h	150	136	210	168	250	200	È	310	232	è	350
	08		28			48			68		88		A8		È	C8		è	E8
9	011	41	051	73	I	111	105	i	151	137	211	169	251	201	É	311	233	é	351
	09		29			49			69		89		A9		É	C9		é	E9
10	012	42	052	74	J	112	106	j	152	138	212	170	252	202	Ê	312	234	ê	352
	0A		2A			4A			6A		8A		AA		Ê	CA		ê	EA
11	013	43	053	75	K	113	107	k	153	139	213	171	253	203	Ë	313	235	ë	353
	0B		2B			4B			6B		8B		AB		Ë	CB		ë	EB
12	014	44	054	76	L	114	108	l	154	140	214	172	254	204	Ì	314	236	ì	354
	0C		2C			4C			6C		8C		AC		Ì	CC		ì	EC
13	015	45	055	77	M	115	109	m	155	141	215	173	255	205	Í	315	237	í	355
	0D		2D			4D			6D		8D		AD		Í	CD		í	ED
14	016	46	056	78	N	116	110	n	156	142	216	174	256	206	Î	316	238	î	356
	0E		2E			4E			6E		8E		AE		Î	CE		î	EE
15	017	47	057	79	O	117	111	o	157	143	217	175	257	207	Ï	317	239	ï	357
	0F		2F			4F			6F		8F		AF		Ï	CF		ï	EF
16	020	48	060	80	P	120	112	p	160	144	220	176	260	208	Ð	320	240	ð	360
	10		30			50			70		90		B0		Ð	D0		ð	F0
17	021	49	061	81	Q	121	113	q	161	145	221	177	261	209	Ñ	321	241	ñ	361
	11		31			51			71		91		B1		Ñ	D1		ñ	F1
18	022	50	062	82	R	122	114	r	162	146	222	178	262	210	Ò	322	242	ò	362
	12		32			52			72		92		B2		Ò	D2		ò	F2
19	023	51	063	83	S	123	115	s	163	147	223	179	263	211	Ó	323	243	ó	363
	13		33			53			73		93		B3		Ó	D3		ó	F3
20	024	52	064	84	T	124	116	t	164	148	224	180	264	212	Ô	324	244	ô	364
	14		34			54			74		94		B4		Ô	D4		ô	F4
21	025	53	065	85	U	125	117	u	165	149	225	181	265	213	Õ	325	245	õ	365
	15		35			55			75		95		B5		Õ	D5		õ	F5
22	026	54	066	86	V	126	118	v	166	150	226	182	266	214	Ö	326	246	ö	366
	16		36			56			76		96		B6		Ö	D6		ö	F6
23	027	55	067	87	W	127	119	w	167	151	227	183	267	215	×	327	247	÷	367
	17		37			57			77		97		B7		×	D7		÷	F7
24	030	56	070	88	X	130	120	x	170	152	230	184	270	216	Ø	330	248	ø	370
	18		38			58			78		98		B8		Ø	D8		ø	F8
25	031	57	071	89	Y	131	121	y	171	153	231	185	271	217	Ù	331	249	ù	371
	19		39			59			79		99		B9		Ù	D9		ù	F9
26	032	58	072	90	Z	132	122	z	172	154	232	186	272	218	Ú	332	250	ú	372
	1A		3A			5A			7A		9A		BA		Ú	DA		ú	FA
27	033	59	073	91	[133	123	{	173	155	233	187	273	219	Û	333	251	û	373
	1B		3B			5B			7B		9B		BB		Û	DB		û	FB
28	034	60	074	92	\	134	124		174	156	234	188	274	220	Ü	334	252	ü	374
	1C		3C			5C			7C		9C		BC		Ü	DC		ü	FC
29	035	61	075	93]	135	125	}	175	157	235	189	275	221	Ý	335	253	ý	375
	1D		3D			5D			7D		9D		BD		Ý	DD		ý	FD
30	036	62	076	94	^	136	126	~	176	158	236	190	276	222	Þ	336	254	þ	376
	1E		3E			5E			7E		9E		BE		Þ	DE		þ	FE
31	037	63	077	95	_	137	127		177	159	237	191	277	223	ß	337	255	ÿ	377
	1F		3F			5F			7F		9F		BF		ß	DF		ÿ	FF

© April 1995, DFL, I5/U10

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;
}
```

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

Etter		
	⋮	
x3010	'a'	str
x3011	'b'	
x3012	'c'	
x3013	0	
x3014	??	
x3015	??	
x3016	??	
x3017	??	
x3018	??	
x3019	??	
x301a	'a'	"abc"
x301b	'b'	
x301c	'c'	
x301d	0	
	⋮	

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 = ''
```

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

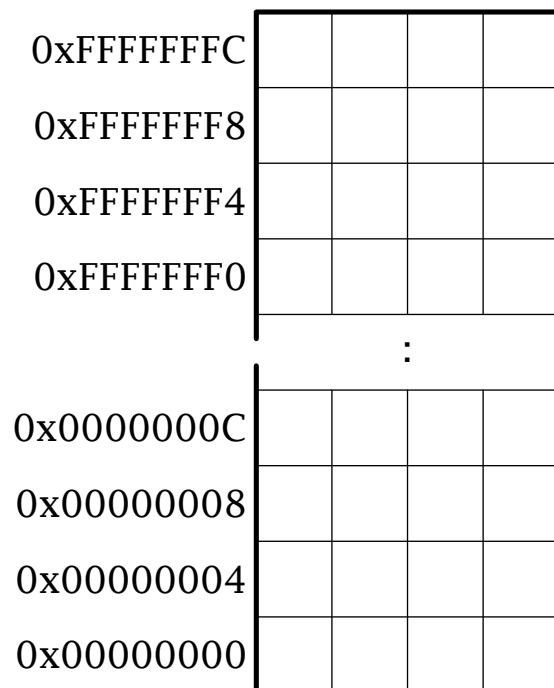
sprintf(str, "...", v1, v2, ...) fungerer som printf men resultatet legges i str i stedet for å skrives ut.

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).

Variable, adresser og pekere

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



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);
}
```

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.

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;
```

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.
```

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);  
}
```


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.

```
#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);
}
```

Legg merke til at både funksjonsdeklarasjonen og kallet er endret!

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);
```

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);
```

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);
```

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)
```

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.)