UNIVERSITETET I OSLO
DET MATEMATISK-NATURVITENSKAPELIGE FAKULTET

Dagens tema

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

INF1070

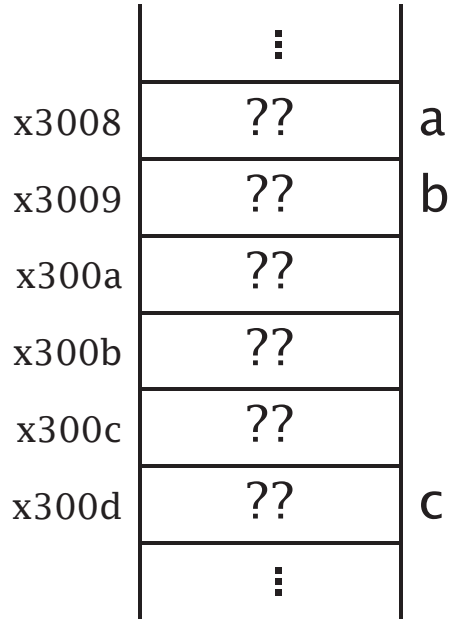
Vektorer

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*.

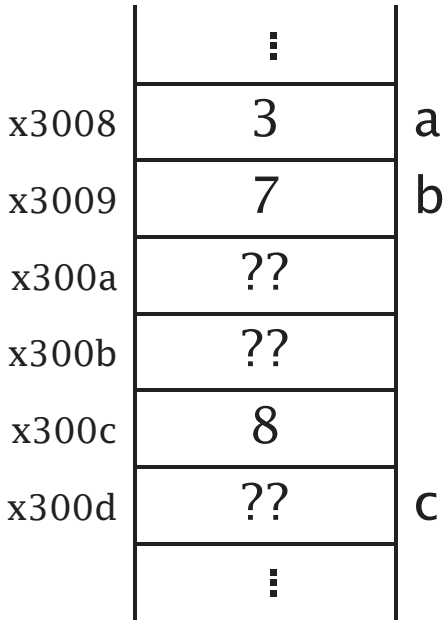
INF1070

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

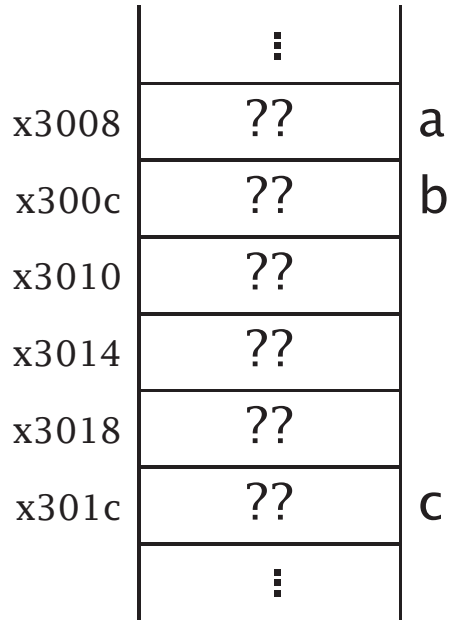
Adressen til vanlige variable er kjent† men adressen til vektorelementer må beregnes. Formelen er

$$Startadresse + Indeks \times Størrelse$$

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.

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.

ISO 8859–1

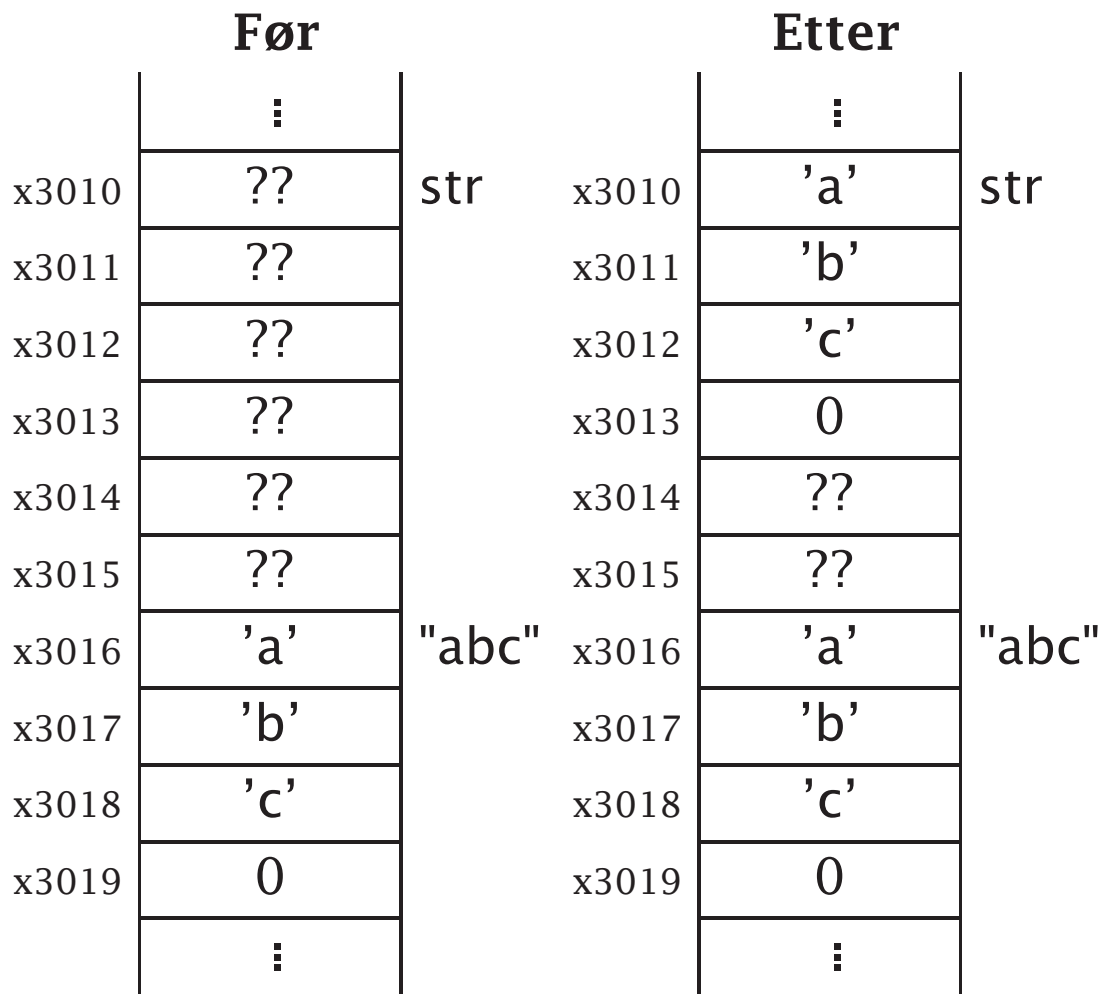
0	000 00	32		040 20	64	@	100 40	96	‘	140 60	128	200 80	160		240 A0	192	À	300 C0	224	à	340 E0
1	001 01	33	!	041 21	65	A	101 41	97	a	141 61	129	201 81	161	¡	241 A1	193	Á	301 C1	225	á	341 E1
2	002 02	34	"	042 22	66	B	102 42	98	b	142 62	130	202 82	162	¢	242 A2	194	Â	302 C2	226	â	342 E2
3	003 03	35	#	043 23	67	C	103 43	99	c	143 63	131	203 83	163	£	243 A3	195	Ã	303 C3	227	ã	343 E3
4	004 04	36	$	044 24	68	D	104 44	100	d	144 64	132	204 84	164	¤	244 A4	196	Ä	304 C4	228	ä	344 E4
5	005 05	37	%	045 25	69	E	105 45	101	e	145 65	133	205 85	165	¥	245 A5	197	Å	305 C5	229	å	345 E5
6	006 06	38	&	046 26	70	F	106 46	102	f	146 66	134	206 86	166	¦	246 A6	198	Æ	306 C6	230	æ	346 E6
7	007 07	39	’	047 27	71	G	107 47	103	g	147 67	135	207 87	167	§	247 A7	199	Ç	307 C7	231	ç	347 E7
8	010 08	40	(	050 28	72	H	110 48	104	h	150 68	136	210 88	168	¨	250 A8	200	È	310 C8	232	è	350 E8
9	011 09	41	)	051 29	73	I	111 49	105	i	151 69	137	211 89	169	©	251 A9	201	É	311 C9	233	é	351 E9
10	012 0A	42	*	052 2A	74	J	112 4A	106	j	152 6A	138	212 8A	170	ª	252 AA	202	Ê	312 CA	234	ê	352 EA
11	013 0B	43	+	053 2B	75	K	113 4B	107	k	153 6B	139	213 8B	171	«	253 AB	203	Ë	313 CB	235	ë	353 EB
12	014 0C	44	,	054 2C	76	L	114 4C	108	l	154 6C	140	214 8C	172	¬	254 AC	204	Ì	314 CC	236	ì	354 EC
13	015 0D	45	–	055 2D	77	M	115 4D	109	m	155 6D	141	215 8D	173	-	255 AD	205	Í	315 CD	237	í	355 ED
14	016 0E	46	.	056 2E	78	N	116 4E	110	n	156 6E	142	216 8E	174	®	256 AE	206	Î	316 CE	238	î	356 EE
15	017 0F	47	/	057 2F	79	O	117 4F	111	o	157 6F	143	217 8F	175	¯	257 AF	207	Ï	317 CF	239	ï	357 EF
16	020 10	48	0	060 30	80	P	120 50	112	p	160 70	144	220 90	176	°	260 B0	208	Đ	320 D0	240	ð	360 F0
17	021 11	49	1	061 31	81	Q	121 51	113	q	161 71	145	221 91	177	±	261 B1	209	Ñ	321 D1	241	ñ	361 F1
18	022 12	50	2	062 32	82	R	122 52	114	r	162 72	146	222 92	178	²	262 B2	210	Ò	322 D2	242	ò	362 F2
19	023 13	51	3	063 33	83	S	123 53	115	s	163 73	147	223 93	179	³	263 B3	211	Ó	323 D3	243	ó	363 F3
20	024 14	52	4	064 34	84	T	124 54	116	t	164 74	148	224 94	180	´	264 B4	212	Ô	324 D4	244	ô	364 F4
21	025 15	53	5	065 35	85	U	125 55	117	u	165 75	149	225 95	181	µ	265 B5	213	Õ	325 D5	245	õ	365 F5
22	026 16	54	6	066 36	86	V	126 56	118	v	166 76	150	226 96	182	¶	266 B6	214	Ö	326 D6	246	ö	366 F6
23	027 17	55	7	067 37	87	W	127 57	119	w	167 77	151	227 97	183	·	267 B7	215	×	327 D7	247	÷	367 F7
24	030 18	56	8	070 38	88	X	130 58	120	x	170 78	152	230 98	184	¸	270 B8	216	Ø	330 D8	248	ø	370 F8
25	031 19	57	9	071 39	89	Y	131 59	121	y	171 79	153	231 99	185	¹	271 B9	217	Ù	331 D9	249	ù	371 F9
26	032 1A	58	:	072 3A	90	Z	132 5A	122	z	172 7A	154	232 9A	186	º	272 BA	218	Ú	332 DA	250	ú	372 FA
27	033 1B	59	;	073 3B	91	[	133 5B	123	{	173 7B	155	233 9B	187	»	273 BB	219	Û	333 DB	251	û	373 FB
28	034 1C	60	<	074 3C	92	\	134 5C	124	|	174 7C	156	234 9C	188	¼	274 BC	220	Ü	334 DC	252	ü	374 FC
29	035 1D	61	=	075 3D	93	]	135 5D	125	}	175 7D	157	235 9D	189	½	275 BD	221	Ý	335 DD	253	ý	375 FD
30	036 1E	62	>	076 3E	94	^	136 5E	126	~	176 7E	158	236 9E	190	¾	276 BE	222	Þ	336 DE	254	þ	376 FE
31	037 1F	63	?	077 3F	95	_	137 5F	127		177 7F	159	237 9F	191	¿	277 BF	223	ß	337 DF	255	ÿ	377 FF

© April 1995, DFL, Ifi/UiO

Kopiering av tekst

Flytting av tekst skjer med standardfunksjonen strcpy:

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



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

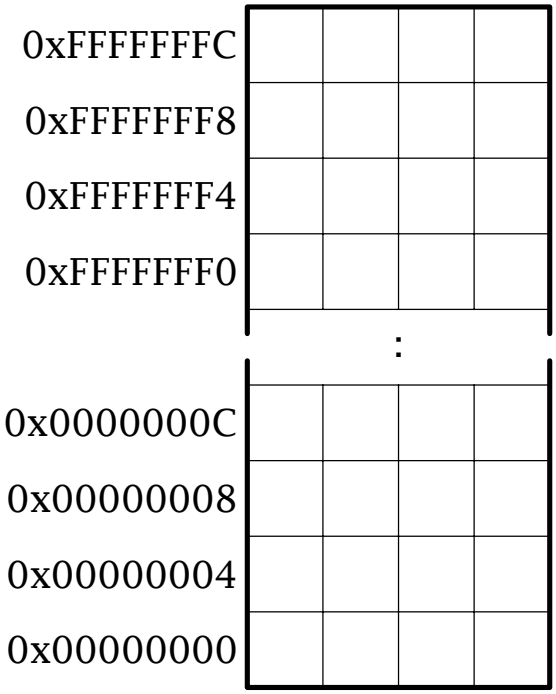
- < 0 om str1 < str2
- 0 om str1 = str2
- > 0 om str1 > str2

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).

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);
}
```

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 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;
```

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:

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

```c
#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;
}
```

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.

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.

```c
#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;
}
```

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);

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)
```

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.)