Dagens tema

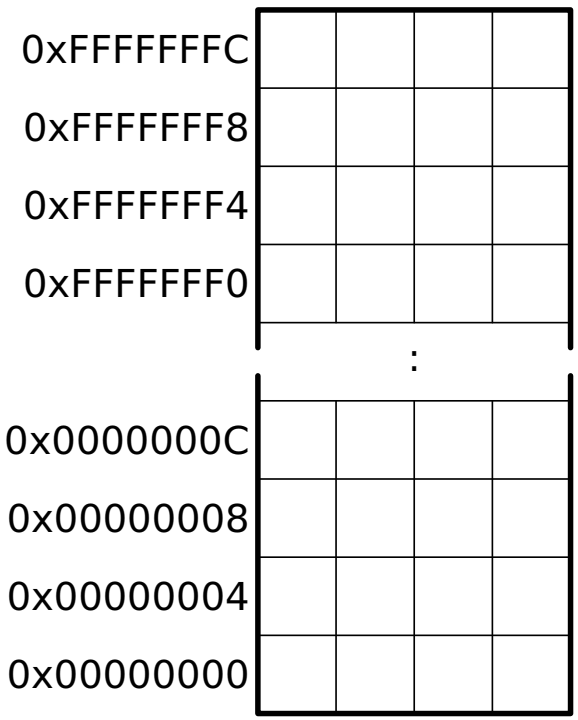
C-programmering

Nøkkelen til å forstå C-programmering ligger i å forstå hvordan minnet brukes.

- Adresser og pekere
- Parametre
- Vektorer (array-er)
- Tekster (string-er)

Variabler, adresser og pekere

Variabler 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 0x%08x ligger a med verdien %d.\n", &a, a);
  printf("I adresse 0x%08x ligger b med verdien %d.\n", &b, b);
  printf("I adresse 0x%08x ligger c med verdien %d.\n", &c, c);
}
```

Innlesning

Til innlesning brukes `scanf`. Første parameter angis hva som skal leses inn: `%c` for tegn, `%d` for heltall og `%f` for flyt-tall.

Legg merke til & foran variabelnavnet; den *må* være der!

La oss kjøre dette programmet:

```
Skriv to tall: 33 21
Summen er 54.
I adresse 0x006009fc ligger a med verdien 33.
I adresse 0x00600a00 ligger b med verdien 21.
I adresse 0x00600a04 ligger c med verdien 54.
```

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.

Adressevariabler

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

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

Her er v en vanlig variabel mens p er en adresse som kan peke på `int`-variabler. (Vi må alltid oppgi hva slags variabler adresser skal peke på.)

Bruk av adressevariabler

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

```
p = &v;
```

Vi kan «følge en adresse» ved å bruke operatoren ∗; da får vi variabelen som adressen 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;
```

```
#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 *adressene* til de to variablene i stedet for verdiene. Adressene overføres som kopier, men vi kan allikevel endre det de peker på.

Vi må endre både deklarasjonen og kallet:

```
#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);
}
```

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 variabler ved å sende over *adressene* til dem. Dette gjøres for eksempel i

```
scanf("%d", &v);
```

Kommentarer

Kommentarer omgis av /∗ og ∗/. De kan stå hvor som helst.

(Mange kompilatorer godtar også
    // . . . ⟨linjeskift⟩
men det er ikke standard i C.)

Logiske («boolean») verdier

C har ingen egen datatype for logiske verdier; i stedet brukes `int` slik:

$= 0$	Usant	(= **false**)
$\neq 0$	Sant	(= **true**)

Lagring av tegn

C har ingen egen type for å lagre *tegn* (som char i Java). I stedet benyttes heltall, oftest **`unsigned char`**.

Hvilken koding som brukes, vil variere fra én maskin til en annen. I den vestlige verden brukes ennå i stor grad **ISO 8859-1** og **-15**, også kjent som **ISO LATIN-1** og **-9**.

(**Unicode** og **UTF-8** er nå i ferd med å overta, og dette vil komplisere programmeringen.)

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

Hvordan lese filer?

Med funksjonen `fopen` kan man åpne filer:

```
#include <stdio.h>

FILE *f = fopen("minfil.txt", "r");
```

(Siste parameter angir at filen skal åpnes for lesing.)

Når man leser fra fil, benyttes **`fscanf`** (i stedet for `scanf`):

```
    fscanf(f, "%d", &v);
```

Vektorer

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

Deklarasjon

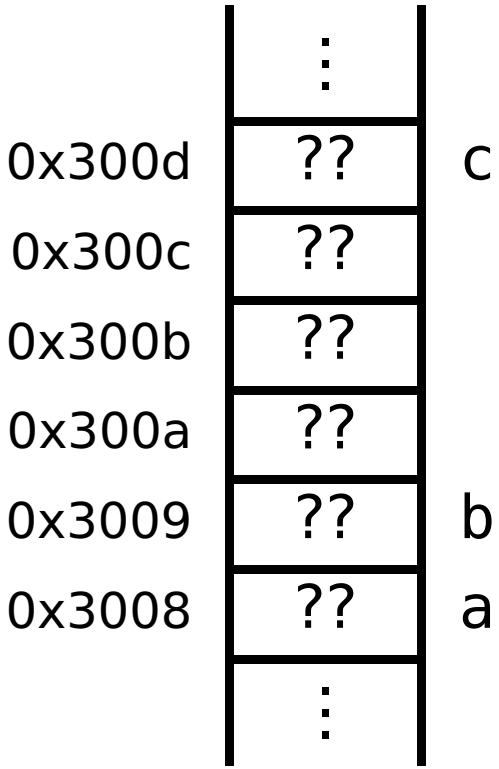
I C deklareres vektorer ved å sette antallet elementer i hakeparenteser etter variabelnavnet:

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

Antallet elementer må være en *konstant*.

Hvordan lagres de i minnet?

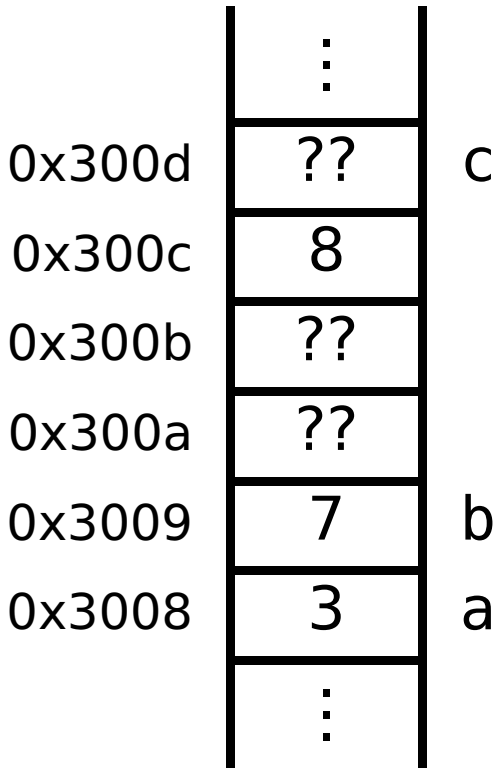
```
char a, b[4], c;
```



Bruk

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

Etter dette er situasjonen:



Beregning av adresse

Man kan regne seg frem til adressen til et vektorelement når man kjenner indeksen og vektorens startadresse; formelen er

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

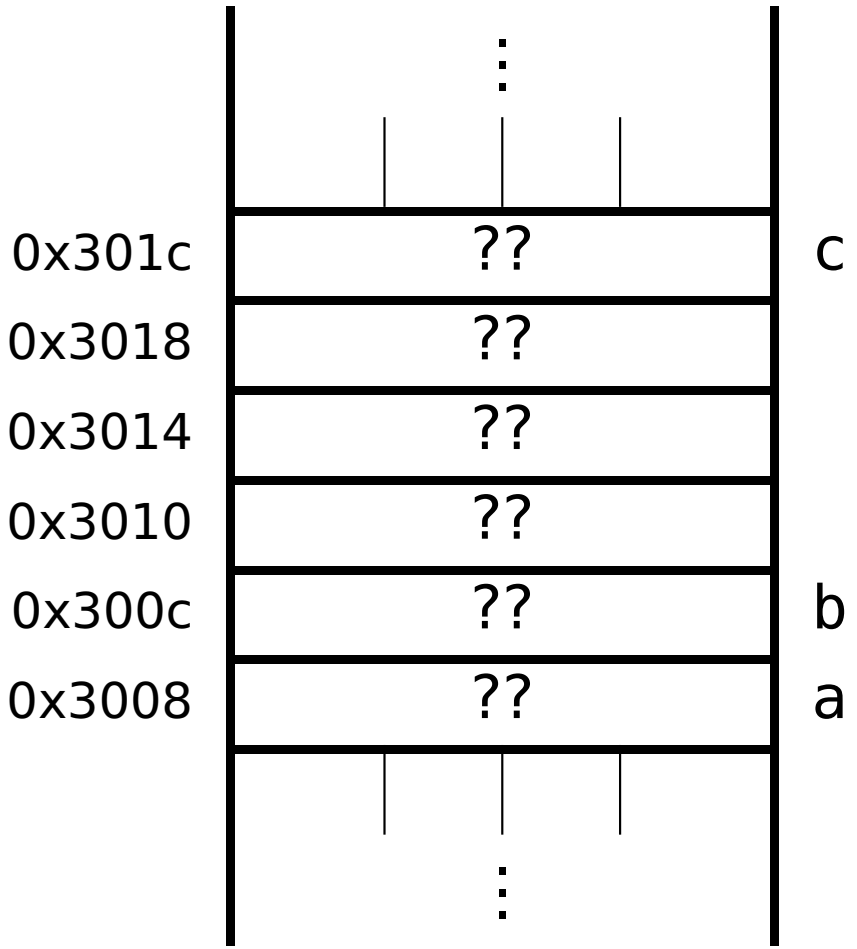
Hva skjer med en ulovlig indeks?

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

Størrelse over 1 byte

Anta at `int` er 4 byte.

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



Flerdimensjonale matriser

I C kan vi ha flerdimensjonale matriser:

```
int a[3][2];
```

Hvordan lagrer man noe slikt i minnet?

Svar

Flerdimensjonale matriser lagres sammenhengende og radvis:

```
#include <stdio.h>

int main(void)
{
  int a[3][2], i, j;

  for (i = 0;  i < 3;  ++i)
    for (j = 0;  j < 2;  ++j)
      printf("A[%d][%d] ligger i adresse 0x%08x.\n",
             i, j, &a[i][j]);
  return 0;
}
```

gir dette svaret:

```
A[0][0] ligger i adresse 0x2bf66160.
A[0][1] ligger i adresse 0x2bf66164.
A[1][0] ligger i adresse 0x2bf66168.
A[1][1] ligger i adresse 0x2bf6616c.
A[2][0] ligger i adresse 0x2bf66170.
A[2][1] ligger i adresse 0x2bf66174.
```

Adresseberegning

Adressen til `A[i1][i2]` beregnes slik:

$$Start + (i_1 \times antall_2 + i_2) \times Størrelse$$

. . . og dette kan utvides til vilkårlig mange dimensjoner.

Tekster

I C lagres tekster som tegnvektorer med en spesiell konvensjon: Etter siste tegn står en byte med verdien 0.[1]

Variabler

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.

[1] En byte med verdien 0 er ikke det samme som sifferet «0»; sifferet «0» er representert av verdien 48; se plansje nr 17.

Kopiering av tekst

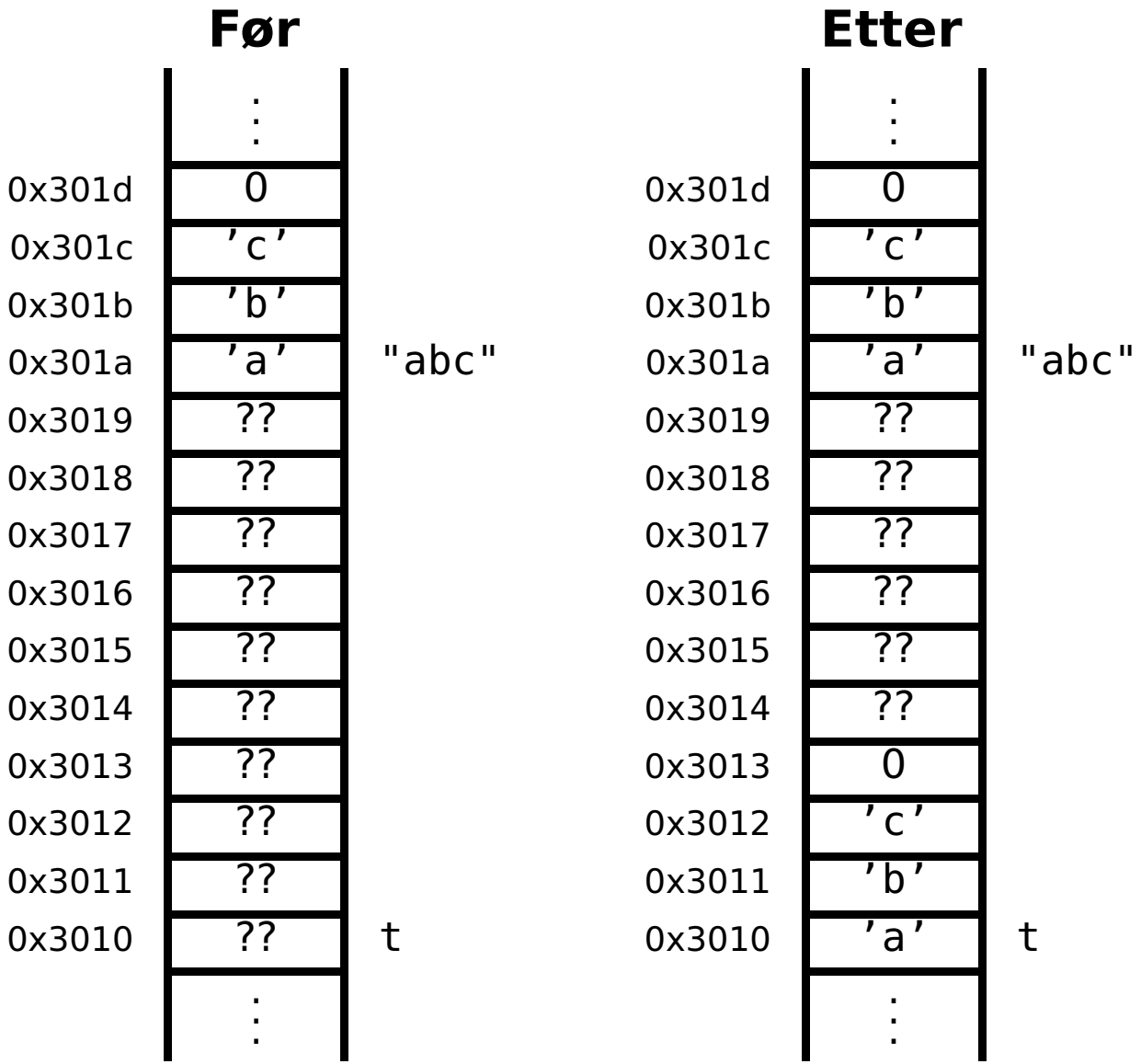
Flytting av tekst skjer med standardfunksjonen `strcpy`:

```
unsigned char *mystrcpy (unsigned char til[], unsigned char fra[])
{
  int i = 0;

  while (1) {
    til[i] = fra[i];
    if (fra[i] == 0) return til;
    ++i;
  }
}
```

Kopiering av tekst

```
strcpy(t,"abc")
```

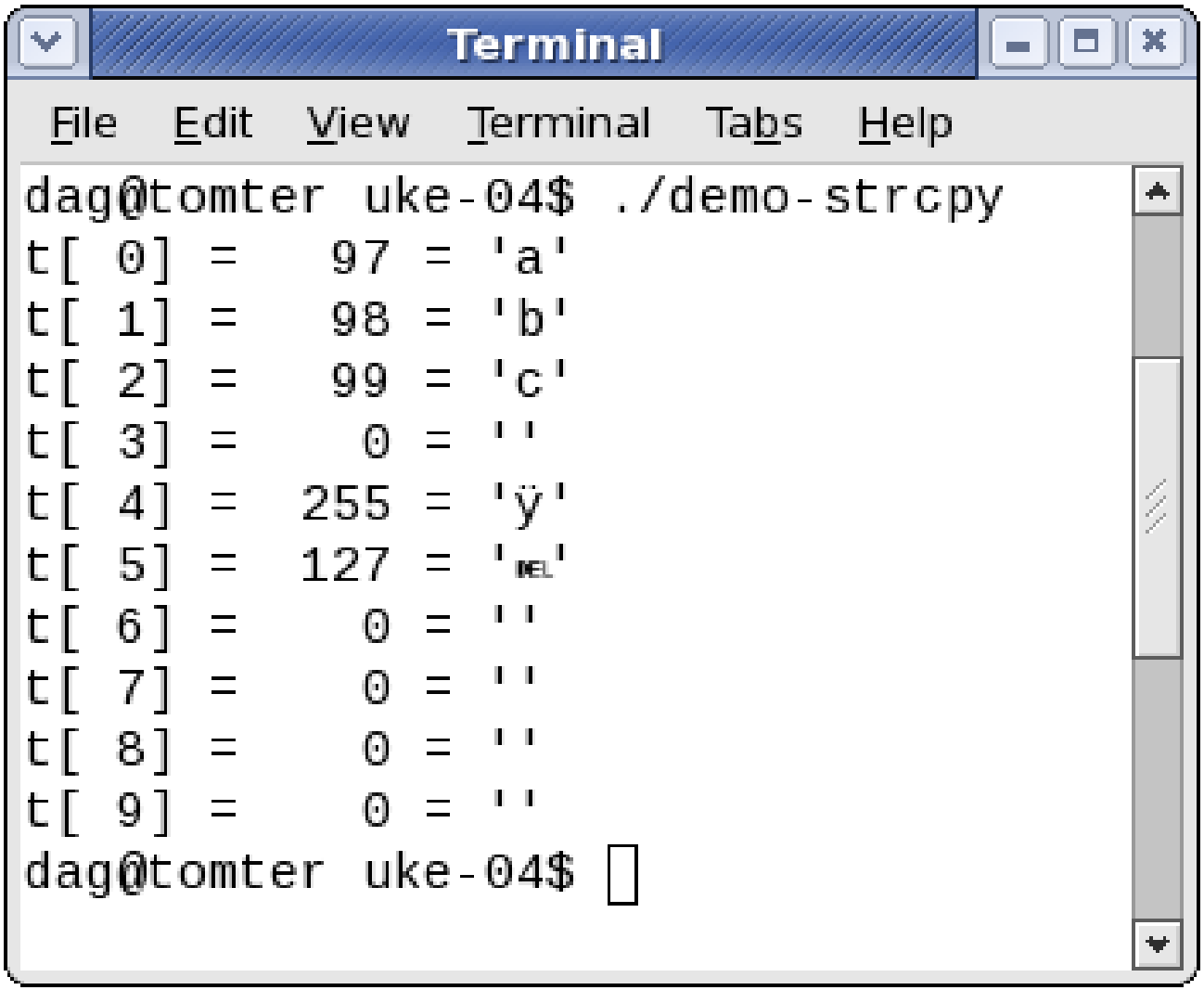


En demonstrasjon

```
#include <stdio.h>


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

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


Terminal
File Edit View Terminal Tabs Help
dag@tomter uke-04$ ./demo-strcpy
t[0] = 97 = 'a'
t[1] = 98 = 'b'
t[2] = 99 = 'c'
t[3] = 0 = ''
t[4] = 255 = 'ÿ'
t[5] = 127 = ''
t[6] = 0 = ''
t[7] = 0 = ''
t[8] = 0 = ''
t[9] = 0 = ''
dag@tomter uke-04$

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 tekstvariabler er vektorer, er det ingen sjekk på plassen. Det er derfor fullt mulig å ødelegge for seg selv (og noen ganger for andre).

Konklusjon om tekster i C

- Programmereren har ansvar for å sette av plass til tekster.
- Nye kopier av tekster oppstår *aldri* automatisk i C.
- Programmereren må sikre at det alltid er nok plass til en tekst.