Dagens tema

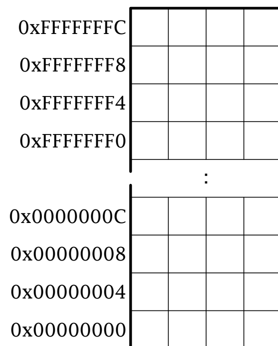
C-programmering

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

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

Variable, adresser og pekere

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



Operatoren &

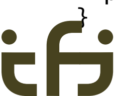
I C kan man få vite i hvilken adresse en variabel ligger ved å bruke operatoren &.

```
#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);
}
```



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: 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.

Adressevariable

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 adresse som kan peke på int-variable. (Vi må alltid oppgi hva slags variable adresser skal peke på.)

Bruk av adressevariable

Vi kan sette adressen til variable 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.

```
#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 *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

$= 0$	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 stort sett ennå ISO 8859-1, også kjent som ISO LATIN-1; om noen år blir det **Unicode**.

Tegnsett

ISO 8859–1

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

© April 1995, DFL, Ifi/UiO

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 variable av samme type hvor man bruker en **indeks** til å skille dem.

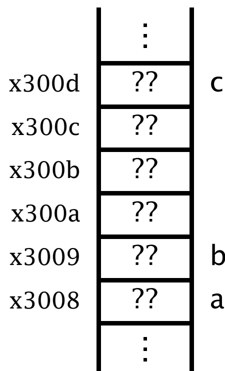
Deklarasjon

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

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

Antallet elementer må være en *konstant*.

char a, b[4], c;



Bruk

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

Etter dette er situasjonen:

Adresse	Innhold	Variabel
	⋮	
x300d	??	c
x300c	8	
x300b	??	
x300a	??	
x3009	7	b
x3008	3	a
	⋮	

Beregning av adresse

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

$$Startadresse + Indeks \times Størrelse$$

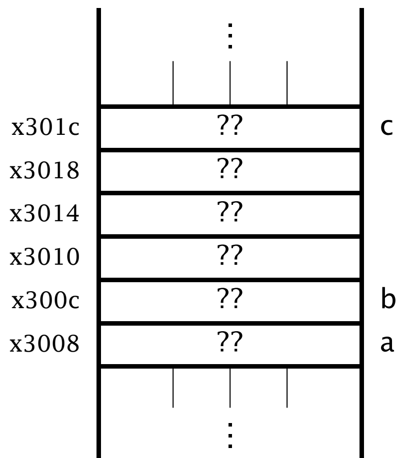
Hva skjer med en ulovlig indeks?

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

Størrelse over 1 byte

Anta at int er 4 byte.

int a, b[4], c;



Tekster

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

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.

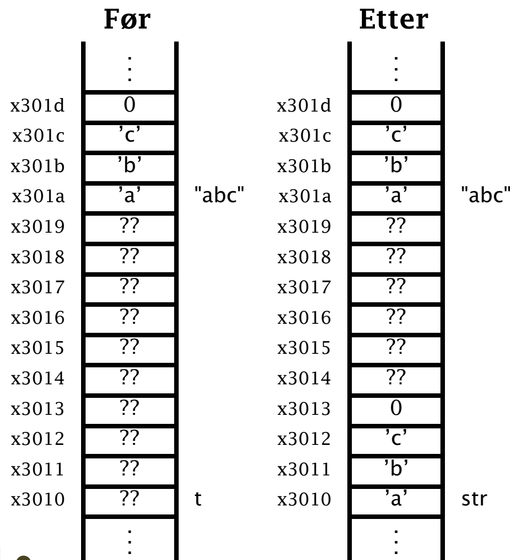
[1] En byte med verdien 0 er ikke det samme som sifferet «0»; sifferet «0» er representert av verdien 48, som vist på lysark nr 17.

Kopiering av tekst

Flytting av tekst skjer med standardfunksjonen strcpy:

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

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


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

En demonstrasjon

```
#include <stdio.h>

int main (void)
{
  unsigned 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;
}
```

Her er utskriften fra kjøringen:

```
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 = ''
```

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).