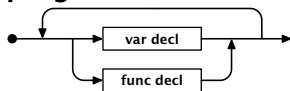


Grammatikk

Vi kan beskrive grammatikken til C_b med *jernbanediagrammer*:

program

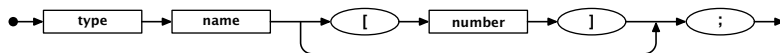


Et program er bare (som i C) en samling deklarasjoner av variabler og funksjoner.

Variabler

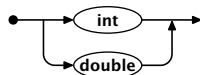
Variabler (enkle eller vektorer) deklarerer slik vi er kjent med fra C:

var decl



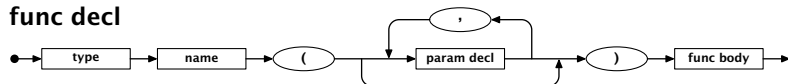
I C_b har vi bare to typer:

type

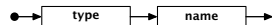


Funksjoner deklarerer som i C og Java:

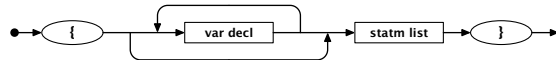
func decl



param decl



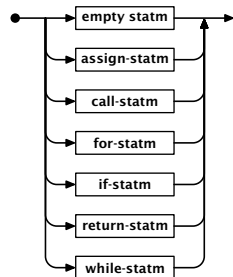
func body



Funksjonen main er hovedprogrammet.

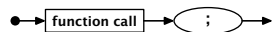
C_b har færre setninger enn C og Java:

statement

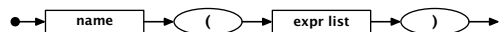


Funksjonskall er som forventet

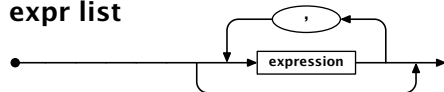
call-stاتم



function call



expr list



Grammatikken forteller om et program er korrekt når det gjelder oppsettet (men det kan allikevel ha logiske feil).

Dette programmet er ikke korrekt:

```
1 int main ()
2 {
3     int x = 5;
4     putint(x+1);    putchar(10);
5 }
```

Kjøring gir

```
$ cflat p1b.cflat
This is the Cb compiler (version 2012-08-16 on Linux)
Parsing...
Cb error in line 3: A semicolonToken expected, but found a assignToken!
```



Eksempel 2

p2.cflat

```
int main ()
{
    int c;  int kind;

    putchar('?');  c = getchar();  /* Read a letter. */
    if (c >= 'a') {
        /* Convert to uppercase (if required) */
        c = c - 32;
    }

    kind = 'C';  /* Initially, assume a consonant. */
    if (c == 'A') { kind = 'V'; }
    else { if (c == 'E') { kind = 'V'; }
          else { if (c == 'I') { kind = 'V'; }
                else { if (c == 'O') { kind = 'V'; }
                      else { if (c == 'U') { kind = 'V'; } } } } }

    putchar(kind);  putchar(10);  exit(0);
}
```



Eksempel 2

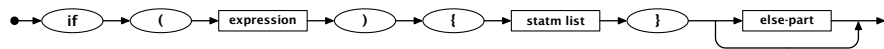
```
$ cflat p2.cflat
This is the Cb compiler (version 2012-08-16 on Linux)
Parsing... checking... generating code... OK
Running gcc -m32 -o p2 p2.s -L. -L/local/share/inf2100 -lcflat
$ ./p2
?f
C
```

Programmet leser en bokstav og avgjør om det en vokal eller konsonant.

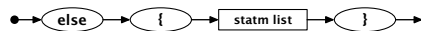
If-setninger

If-setninger tester på 0 (*usant*) og $\neq 0$ (*sant*). Legg merke til at det alltid skal være krøllparenteser.

if-statm

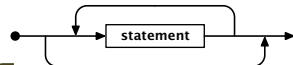


else-part



Setningslister inneholder 0 eller flere setninger:

statm list



Kommentarer

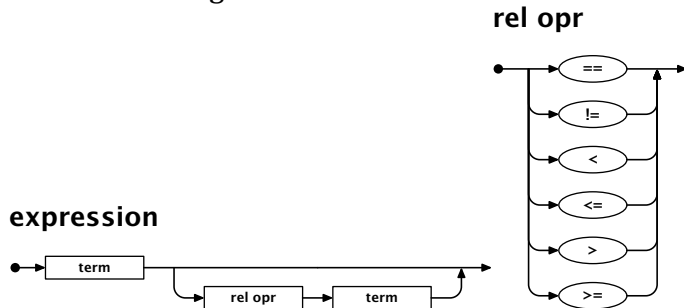
Det finnes to former for kommentarer:

- `/* ... */`
kan gå over flere linjer.
- `# ...`
er en ren kommentarlinje (om # står helt til venstre)

(Disse inngår ikke blant jernbandediagrammene – de håndteres på et annet plan.)

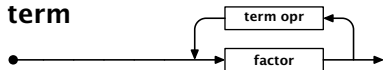
Uttrykk

Grammatikken for <expression> er uventet komplisert, men det er en grunn til det.

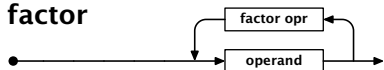


Syntaks for (expression)

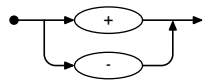
term



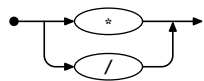
factor



term opr

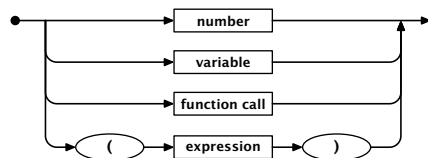


factor opr

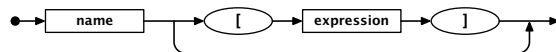


Syntaks for (operand) og (variable)

operand

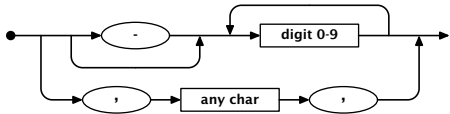


variable



Syntaks for (number)

number



Et tegn (som 'a') er bare en annen notasjon for et tall (97).

ISO 8859-1

000	001	002	003	004	005	006	007	008	009	010	011	012	013	014	015	016	017	018	019	020	021	022	023	024	025	026	027	028	029	030	031	032	033	034	035	036	037	038	039	040	041	042	043	044	045	046	047	048	049	050	051	052	053	054	055	056	057	058	059	060	061	062	063	064	065	066	067	068	069	070	071	072	073	074	075	076	077	078	079	080	081	082	083	084	085	086	087	088	089	090	091	092	093	094	095	096	097	098	099	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255
!	"	#	\$	%	&	'	()	*	+	,	-	.	/	0	1	2	3	4	5	6	7	8	9	:	<	=	>	?	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	{		}	~	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ																																																																																																			

Eksempel 3

p3.cflat

```
int LF;

int gcd (int a, int b)
{
    while (a != b) {
        if (a < b) {
            b = b-a;
        } else {
            a = a-b;
        }
    }
    return a;
}

int main ()
{
    int v1; int v2; int res;

    LF = 10;
    putchar('?');
    v1 = getint(); v2 = getint();
    res = gcd(v1,v2);
    putint(res); putchar(LF);
}
```



Programmet finner *største felles divisor*:

```
$ cflat p3.cflat
This is the Cb compiler (version 2012-08-16 on Linux)
Parsing... checking... generating code... OK
Running gcc -m32 -o p3 p3.s -L. -L/local/share/inf2100 -lcflat
$ ./p3
?52 221
13
```

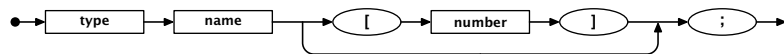
Biblioteket

C**b** kjenner til disse syv funksjonene:

<code>exit(<i>status</i>)</code>	Avslutter programmet
<code>getchar()</code>	Leser et tegn fra tastaturet
<code>getdouble()</code>	Leser et flyttall fra tastaturet
<code>getint()</code>	Leser et heltall fra tastaturet
<code>putchar(<i>n</i>)</code>	Skriver ut et tegn på skjermen
<code>putdouble(<i>x</i>)</code>	Skriver ut et flyttall på skjermen
<code>putint(<i>n</i>)</code>	Skriver ut et heltall på skjermen

Vektorer

var decl



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

deklarerer a med elementene a[0], a[1] og a[2]. Det er ingen sjekk på grensene.

Eksempel 4

primes.cflat, del 1

```
# Program 'primes'
# -----
#
# Finds all prime numbers up to 1000 (using the technique called
# "the sieve of Eratosthenes") and prints them nicely formatted.

int prime[1001]; /* The sieve */
int LF;          /* LF */

int find_primes ()
{
    /* Remove all non-primes from the sieve: */

    int i1; int i2;
    for (i1 = 2; i1 <= 1000; i1 = i1+1) {
        for (i2 = 2*i1; i2 <= 1000; i2 = i2+i1) {
            prime[i2] = 0;
        }
    }
}
```



primes.cflat, del 2

```
int mod (int a, int b)
{
    /* Computes a%b. */
    return a - a/b*b;
}
```

primes.cflat, del 3

```
int n_chars (int a)
{
    /* How many positions are needed to print 'a'? */

    if (a < 0) { return 1+n_chars(0-a); }
    if (a <= 9) { return 1; }
    return n_chars(a/10)+1;
}
```

primes.cflat, del 4

```
int pn (int v, int w)
{
    /* Does a 'printf("%*d", w, v)'. */
    int i;
    for (i = n_chars(v)+1; i <= w; i = i+1) { putchar(' '); }
    putint(v);
}
```

primes.cflat, del 5

```
int and (int a, int b)
{
    /* Compute a && b . */

    if (a) { return b; }
    else  { return 0; }
}
```


primes.cflat, del 6

```
int print_primes ()
{
    /* Print the primes, 10 on each line. */

    int n_printed; int i;
    n_printed = 0;
    for (i = 1; i <= 1000; i = i + 1) {
        if (prime[i]) {
            if (and(mod(n_printed,10)==0, n_printed>0)) { putchar(LF); }
            putchar(' '); pn(i,3); n_printed = n_printed+1;
        }
    }
    putchar(LF);
}
```

primes.cflat, del 7

```
int main ()
{
    int i;
    LF = 10;

    /* Initialize the sieve by assuming all numbers >1 to be primes: */
    prime[1] = 0;
    for (i=2; i<=1000; i=i+1) { prime[i] = 1; }

    /* Find and print the primes: */
    find_primes(); print_primes();
    return 0;
}
```

Eksempel 4

```
$ cflat primes.cflat
This is the Cb compiler (version 2012-08-16 on Linux)
Parsing... checking... generating code... OK
Running gcc -m32 -o primes primes.s -L. -L/local/share/inf2100 -lcflat
$ ./primes
 2   3   5   7  11  13  17  19  23  29
31  37  41  43  47  53  59  61  67  71
73  79  83  89  97 101 103 107 109 113
127 131 137 139 149 151 157 163 167 173
179 181 191 193 197 199 211 223 227 229
233 239 241 251 257 263 269 271 277 281
283 293 307 311 313 317 331 337 347 349
353 359 367 373 379 383 389 397 401 409
419 421 431 433 439 443 449 457 461 463
467 479 487 491 499 503 509 521 523 541
547 557 563 569 571 577 587 593 599 601
607 613 617 619 631 641 643 647 653 659
661 673 677 683 691 701 709 719 727 733
739 743 751 757 761 769 773 787 797 809
811 821 823 827 829 839 853 857 859 863
877 881 883 887 907 911 919 929 937 941
947 953 967 971 977 983 991 997
```



Oppsummering

Cb er et meget enkelt programmeringsspråk.

- Det bør være enkelt å lære.
- Ikke alt man forventer, er lov. Sjekk grammatikken!