Velkommen til INF2100

Jeg er *Dag Langmyhr* (dag@ifi.uio.no).

Dagens tema:

- Hva går kurset ut på?
  - Bakgrunn for kurset
  - Hvordan gjennomføres kurset?
  - Hvordan får man det godkjent?
- Programmeringsspråket **C♭**
  - En kort oversikt
  - Syntaks
  - Fire eksempler

En spesiell hilsen

For første gang går dette kurset også ved Universitetet i Tromsø med betegnelsen **UIT-INF-2401**. Et spesielt velkommen til dere i Tromsø.

Bakgrunnen for INF2100

I begynnerkursene har dere lært å programmere, men bare små programmer (< 500 linjer).

Hensikten med INF2100 er

- å gi mer programmeringstrening
- forstå mekanismene man trenger i større programmer (som objektorientering og moduler)
- gi trening i å lese og forstå andres programkode

Prosjektet

Prosjektet er valgt fordi det har en nytteverdi i seg selv:

Prosjekt:

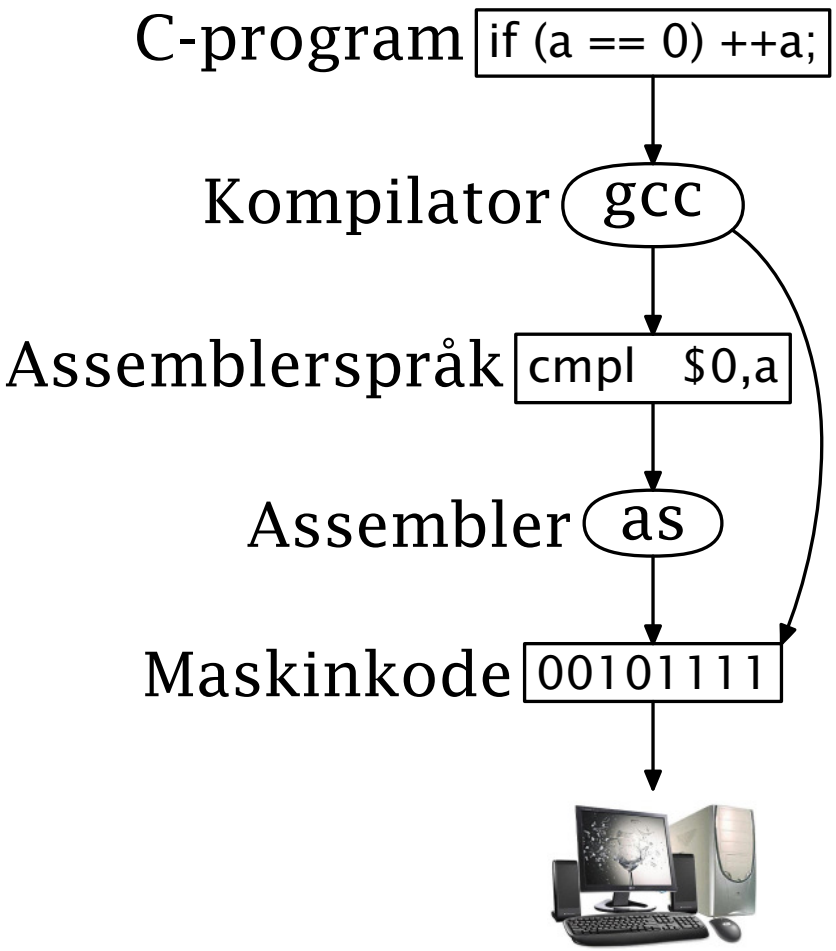
Skriv en kompilator for programmeringsspråket **C♭**.

Dette gir

- forståelse for hvordan en kompilator fungerer
- kjennskap til maskin- og assemblerspråk
- eksempel på hvorledes et programmeringsspråk er definert og bygget opp

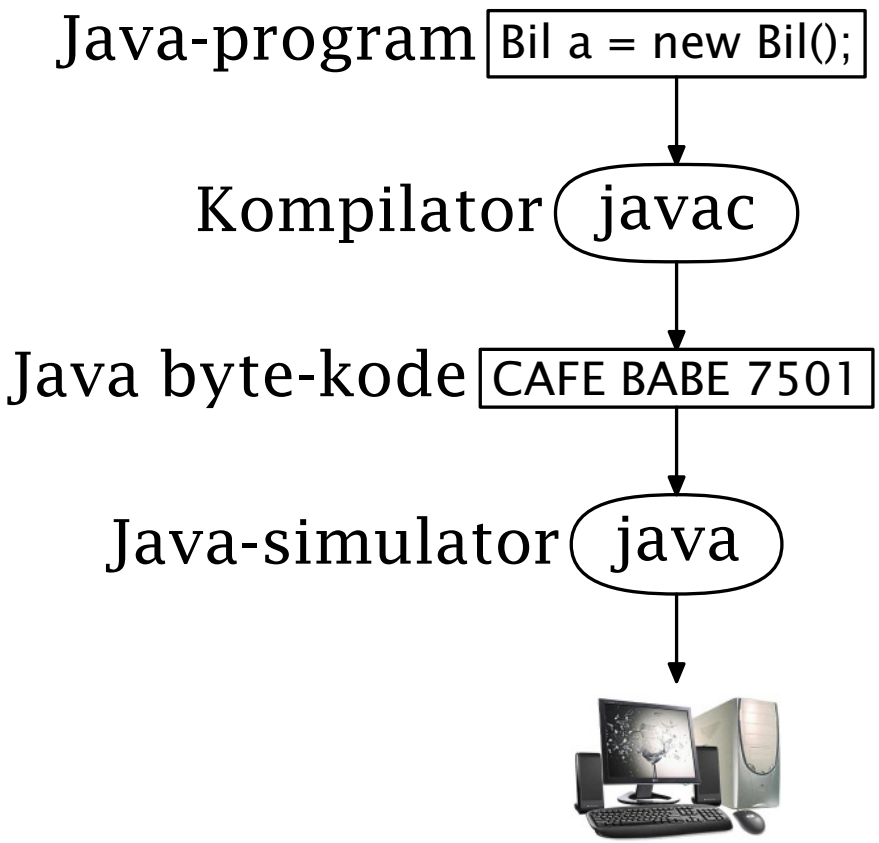
Hva gjør en kompilator?

En kompilator oversetter et program til en annen kode, oftest maskin- eller assemblerkode.

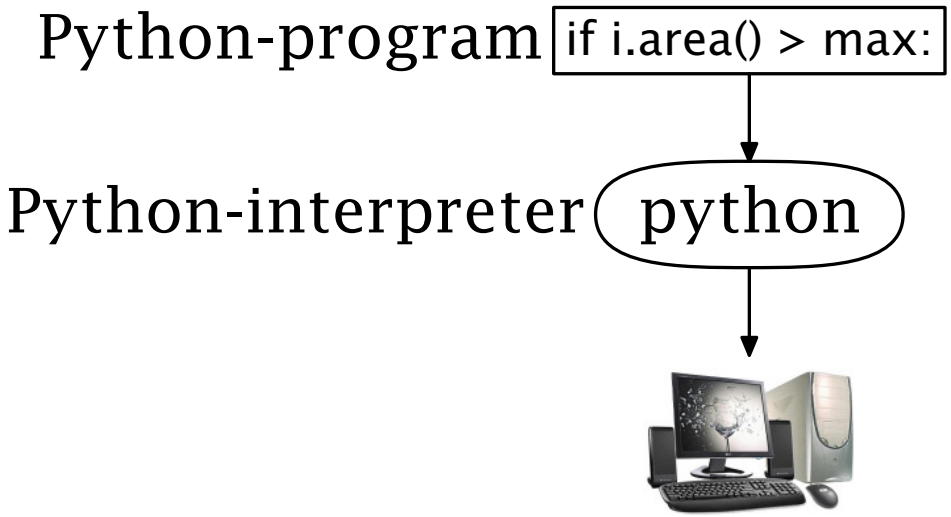


Noen kompilatorer oversetter til en egen kode som *tolkes* av en interpreter.

(Det finnes også Java-kompilatorer som lager maskinkode, men de er ikke så vanlige.)



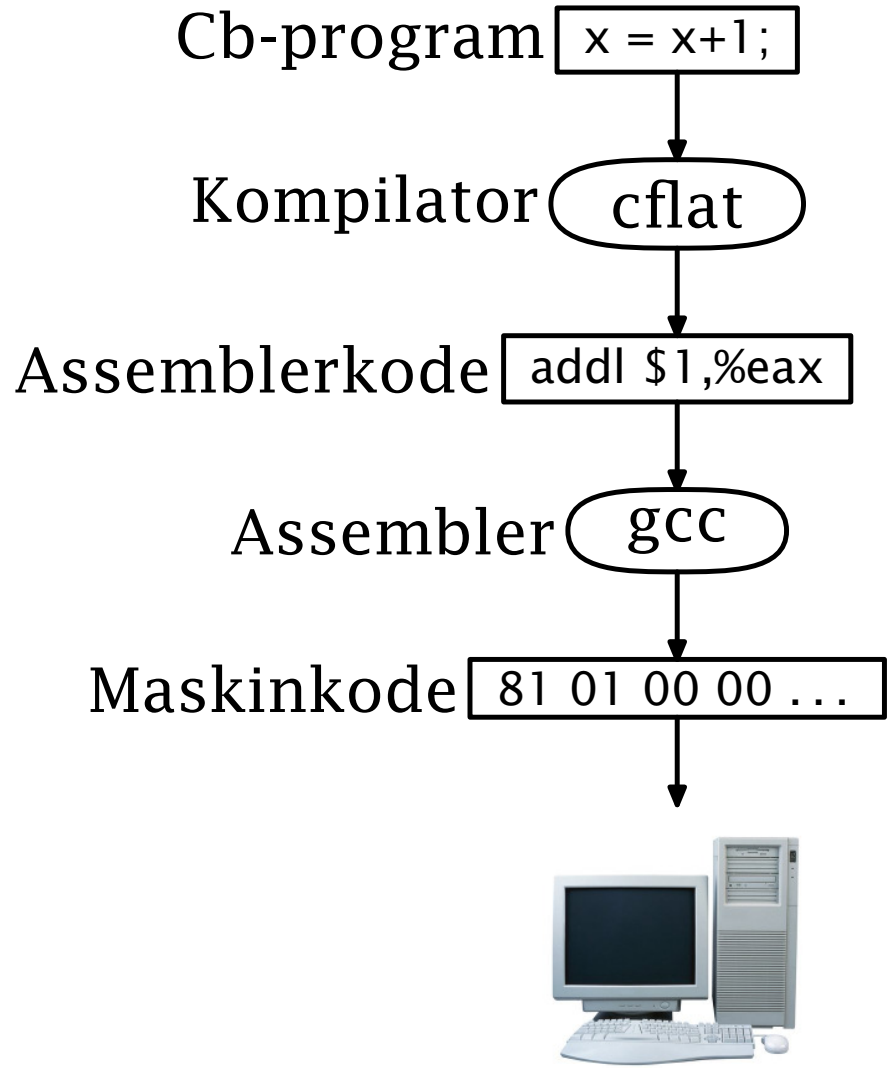
Noen interpretere leser kildekoden direkte (men vil oversette den til en passende form internt).



C♭-kompilatoren

Opplegget for C♭ ligner mye på det for C.

Oppgaven deres er å skrive denne kompilatoren.



Ulike programmeringsspråk

I dette kurset skal vi innom flere språk:

- **Java**, som benyttes til all implementasjon.
- **C♭** («C flat»), som er målet for prosjektet.
- **x86-maskinkode** til datamaskinen vår
- **assemblerspråket** til x86.

Oppbyggingen av kurset

Kurset har syv hovedkomponenter:

- **Forelesningene** holdes annenhver uke (sånn omtrent); se http://www.uio.no/studier/emner/matnat/ifi/INF2100/h13/timeplan/. Forelesningene blir lagt ut som podcast etterpå.
- **Gruppeøvelser** 2 timer hver uke.
- **Gruppearbeid** for å løse oppgavene. Man jobber to og to (eller alene om man vil).
- **Kompendiet** gir en grundig innføring i C♭ og prosjektet.
- **Basiskoden** er utgangspunktet for programmeringen deres. Den skal bare utvides, ikke endres!
- **Nettsidene** er også en viktig informasjonskanal.
- **Nettforumet** kan benyttes til informasjon og diskusjon.

Basiskoden

Det er flere grunner til å dere skal basere prosjektet på en utdelt basiskode:

- Det gjør det lettere for dere å komme i gang.
- Det er nyttig å bli vant til å lese andres kode.
- Jeg har bevisst benyttet noen uvante kodeteknikker.
- Uten basiskode ville gruppelærerne fått en uoverkommelig jobb med rettingen.

Godkjenning av kurset

Kurset har ikke karakterer, bare bestått/ikke bestått.

Det er tre obligatoriske oppgaver. Når de er godkjent, er kurset godkjent.

Men . . .

Mot slutten av semesteret vil noen bli plukket ut til en samtale om programmet de har laget. Dette kan man stryke på!

Siden man normalt jobber i lag, forventes at

- begge har kjennskap til hovedstrukturen i programmet
- begge kan identifisere sin del av programmet (som skal være rundt halvparten) der de kan forklare nøye hvorfor koden er blitt slik den er.

Samarbeid og fusk

Samarbeid og utveksling av ideer er bra!

Kopiering og fusk er ikke bra!

Gode råd for samarbeid

- Snakk gjerne med andre om ideer.
- Kopier aldri andres kode.

Ulike programmeringshjelpemidler

Når man skal programmere et større prosjekt, trenger man hjelpemidler:

- et kompileringssystem (à la make eller ant)
- et dokumentasjonsopplegg (à la JavaDoc)
- et versjonskontrollsystem (à la Subversion)
- ...

Dette blir tatt opp i kurset (når dere er motivert for å høre om dem ☺).

En siste ting

Det er ingen krav til utviklingsverktøy. Hvis du vil bruke et program som Eclipse, er det helt OK.

Det finnes åpne kodebanker med gode utviklingsverktøy, som GitHub. Det er ikke lov å bruke disse hvis kildekoden deres blir tilgjengelig for alle.

Programmeringsspråket C♭

Dette språket («C flat») er en meget forenklet utgave av C spesiallaget til dette kurset.

p0.cflat

```
int main ()
{
   putchar('!');   putchar(10);
}
```

p0.cflat

Programmet kompileres og kjøres med kommandoene

```
$ cflat p0.cflat
This is the Cb compiler (version 2013-07-01 on Linux)
Parsing... checking... generating code... OK
Running gcc -m32 -o p0 p0.s -L. -L/local/share/inf2100 -lcflat
$ ./p0
!
```

eller

```
$ java -jar /local/opt/inf2100/Cflat.jar p0.cflat
This is the Cb compiler (version 2013-07-01 on Linux)
Parsing... checking... generating code... OK
Running gcc -m32 -o p0 p0.s -L. -L/local/share/inf2100 -lcflat
$ ./p0
!
```

Siden C♭ er forenklet C, kan vi også bruke C-kompilatoren til å kompilere koden (om vi tar med en biblioteksfil):

```
$ more p1.cflat
int main ()
{
   int x;   x = 5;
   putint(x+1);   putchar(10);
}
$ gcc -m32 -x c -o p1 p1.cflat -L/local/share/inf2100 -lcflat
$ ./p1
6
```

Grammatikk

Vi kan beskrive grammatikken til C♭ med *jernbanediagrammer*:

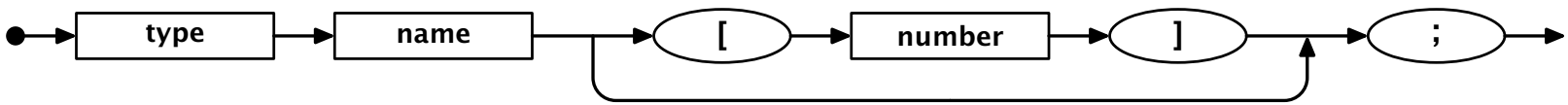
program

Et program er bare en samling deklarasjoner av variabler og funksjoner.

Variabler

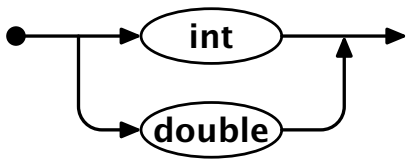
Variabler (enkle eller vektorer) deklareres slik vi er kjent med fra C:

var decl



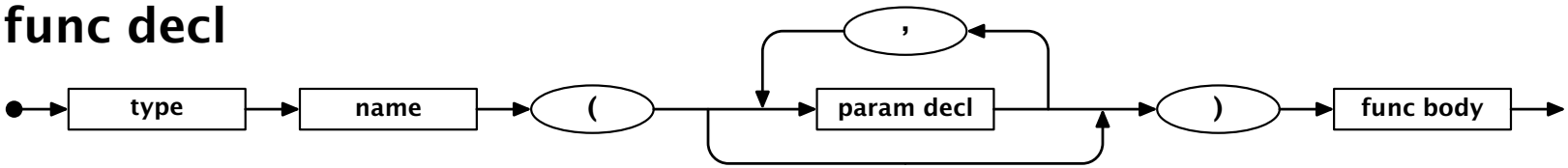
I C♭ har vi bare to typer:

type

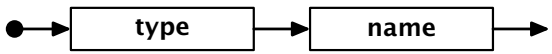


Funksjoner deklareres som i C og Java:

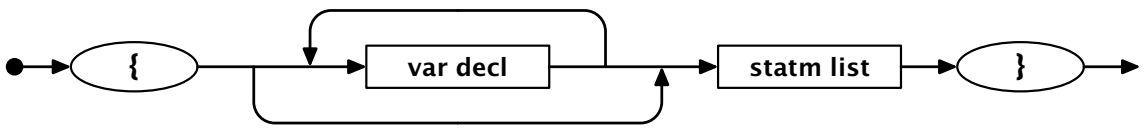
func decl



param decl



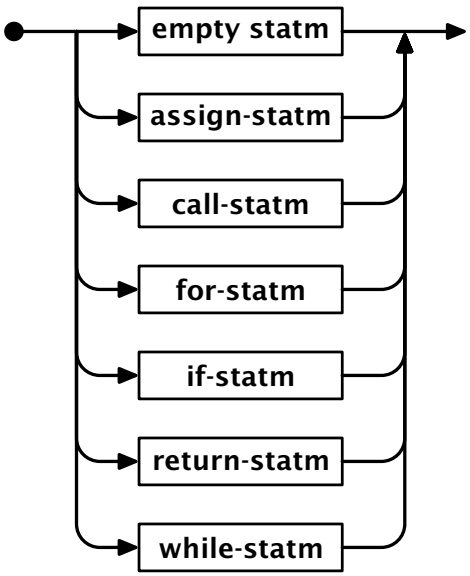
func body



Funksjonen main er hovedprogrammet.

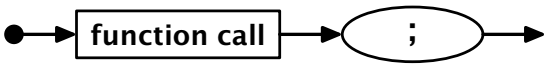
C♭ har færre setninger enn C og Java:

statement

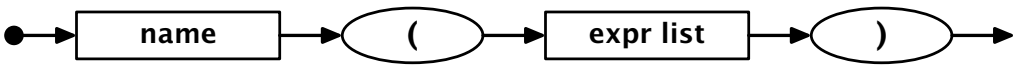


Funksjonskall er som forventet

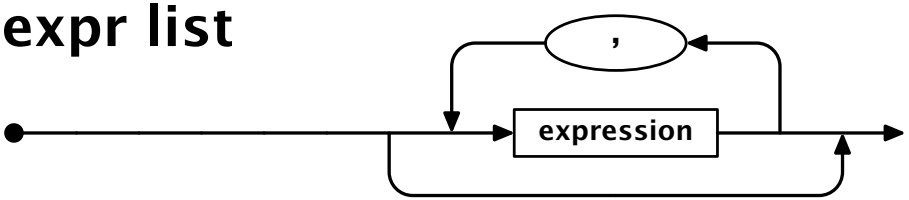
call-statm



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 2013-07-01 on Linux)
Parsing...
Cb error in line 3: A semicolonToken expected, but found a assignToken!
```

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

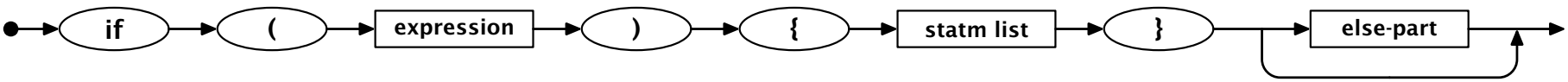
```
$ cflat p2.cflat
This is the Cb compiler (version 2013-07-01 on Linux)
Parsing... checking... generating code... OK
Running gcc -m32 -o p2 p2.s -L. -L/local/share/inf2100 -lcflat
$ ./p2
?f
C
```

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

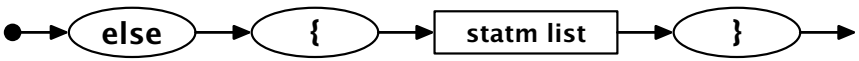
If-setninger

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

if-statm

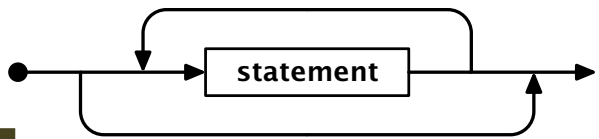


else-part



Setningslister inneholder 0 eller flere setninger:

statm list



Kommentarer

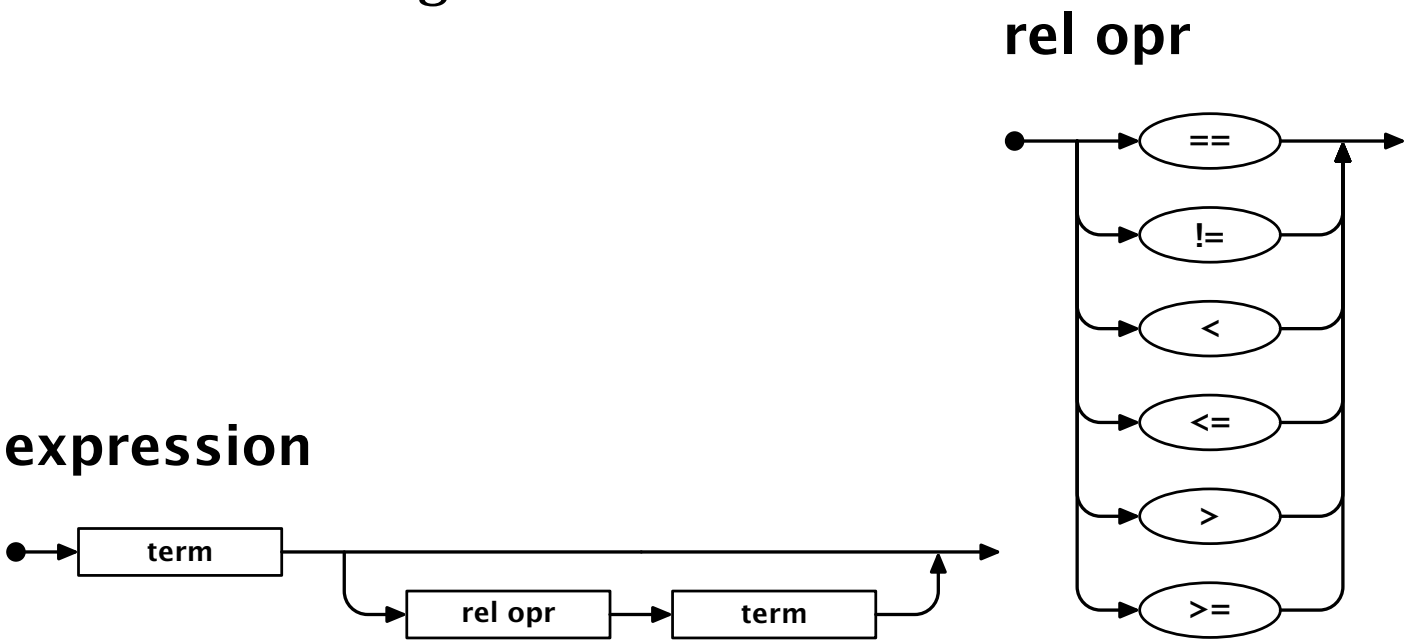
Det finnes to former for kommentarer:

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

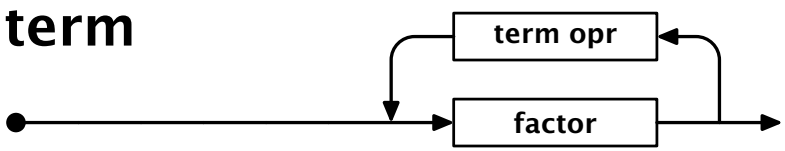
(Disse inngår ikke blant jernbanediagrammene – de håndteres på et annet plan.)

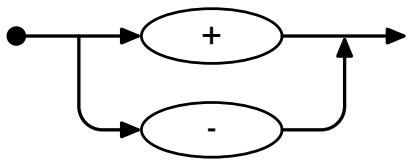
Uttrykk

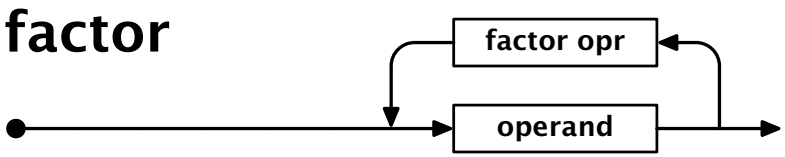
Grammatikken for ⟨expression⟩ er uventet komplisert, men det er en grunn til det.

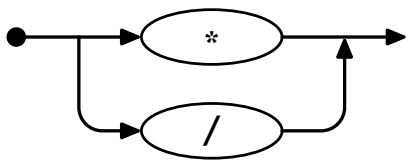


Syntaks for ⟨expression⟩

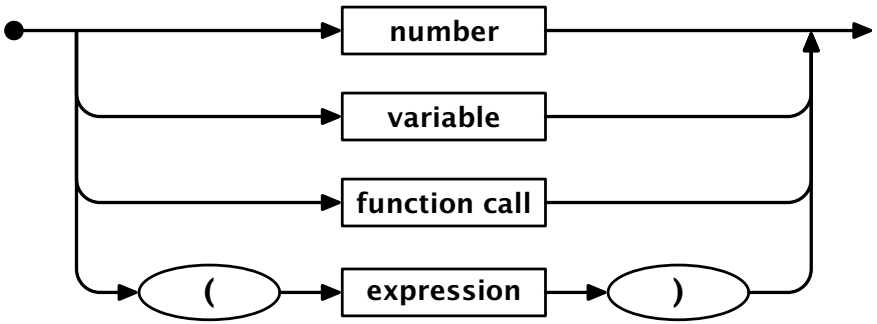




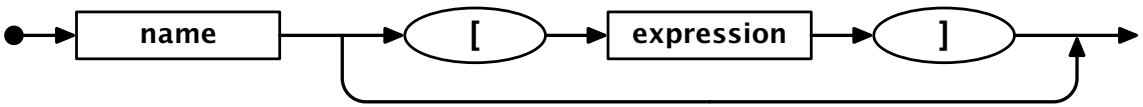


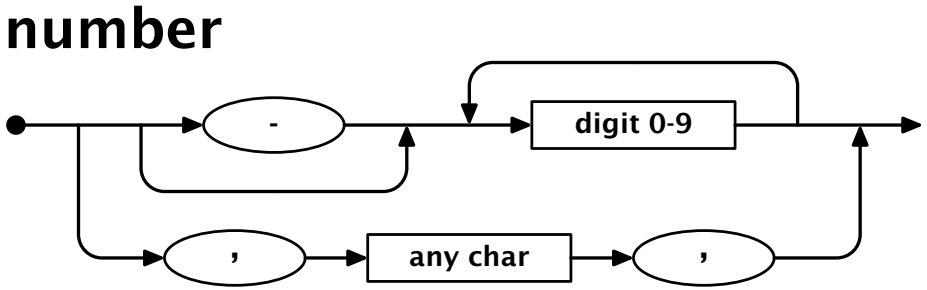


operand



variable





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

I dette kurset antar vi at alle tegn er ASCII-tegn, dvs har representasjon 0–127.

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);
}
```

Eksempel 3

Programmet finner *største felles divisor*:

```
$ cflat p3.cflat
This is the Cb compiler (version 2013-07-01 on Linux)
Parsing... checking... generating code... OK
Running gcc -m32 -o p3 p3.s -L. -L/local/share/inf2100 -lcflat
$ ./p3
?52 221
13
```

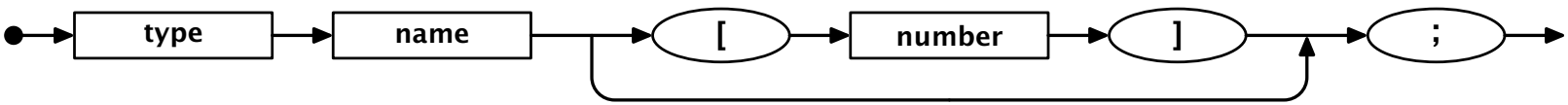
Biblioteket

C♭ kjenner til disse syv funksjonene:

exit(*status*)	Avslutter programmet
getchar()	Leser et tegn fra tastaturet
getdouble()	Leser et flyttall fra tastaturet
getint()	Leser et heltall fra tastaturet
putchar(*n*)	Skriver ut et tegn på skjermen
putdouble(*x*)	Skriver ut et flyttall på skjermen
putint(*n*)	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.

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 2013-07-01 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

C♭ er et meget enkelt programmeringsspråk.

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