

Dagens tema: Resten av det dere trenger til del 1

- Uttrykk
- Typer
- Testutskrifter
- Programmeringsstil
- Dokumentasjon
- 12 gode råd

Prosjektet

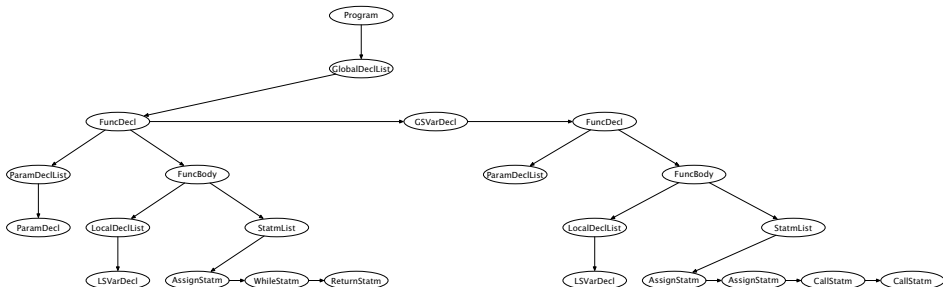
Utifra dette C_b-programmet:

```
int pot2 (int x)
{
    int p2;    p2 = 1;
    while (2*p2 <= x) { p2 = 2*p2; }
    return p2;
}

int x;

int main ()
{
    int v;    v = getint();
    x = pot2(v);    putint(x);    putchar(10);
}
```

... skal vi bygge opp dette treet (som her er klippet):



Koblingen mellom grammatikken og klasser

Normalt skal det være én klasse for hvert metasymbol (definisjon) i grammatikken, men det er lov å avvike fra dette.

- Nye klasser som **Declaration** og **DeclList** er nyttig.
Grunn: Vi ønsker felles data eller metoder for klasser som hører naturlig sammen.
- Vi kan implementere **VarDecl** som abstrakt med fem subklasser: **GlobalArrayDecl**, **GlobalSimpleVarDecl**, **LocalArrayDecl**, **LocalSimpleVarDecl** og **ParamDecl**.
Grunn: Noen metoder vil være forskjellige, f eks for kodegenerering.

Hint

Hvis du stadig programmerer «Hvis det er den varianten, gjør jeg det, og hvis ...», kan subklasse med virtuell metode være løsningen.



Uttrykk

Et uttrykk har én eller to termer:

expression



Hvordan lagre uttrykk

```
class Expression extends Operand {
    Expression nextExpr = null;
    Term firstTerm = new Term(), secondTerm = null;
    Operator relOp = null;
    boolean innerExpr = false;

    @Override void parse() {
        Log.enterParser("<expression>");

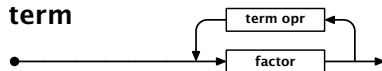
        firstTerm.parse();
        if (Token.isRelOperator(Scanner.curToken)) {
            relOp = new RelOperator();
            relOp.parse();
            secondTerm = new Term();
            secondTerm.parse();
        }

        Log.leaveParser("</expression>");
    }
}
```

Termer

En term har én eller flere faktorer:

term



Implementasjonen av dette må dere finne ut av selv.

```
[SyntaxUnit]
ExprList
[Operand]
  Expression
  FunctionCall
  Number
  Variable
[Operator]
  RelOperator
Term
```

- Disse klassene (og de du lager selv) brukes til å lagre uttrykk.
- Legg merke til at `<expression>` er en subklasse av `<operand>`. Det er fordi vi kan ha indre uttrykk i parenteser (indikert av variabelen `Expression.inner`).

Typer

Modulen Types inneholder klasser og variabler for å representere typer. Basisklassen er Type:

Type.java

```
package no.uio.ifi.cflat.types;

public abstract class Type {
    public abstract int size();
    public abstract String typeName();

    public abstract void checkSameType(int lineNum, Type otherType, String what);
    public abstract void checkType(int lineNum, Type correctType, String what);
    public void genJumpIfZero(String jumpLabel) {}
}
```

Enkle type (dvs int og double) er instanser av BasicType.

BasicType.java

```
package no.uio.ifi.cflat.types;

public abstract class BasicType extends Type {
    @Override public void checkSameType(int lineNum, Type otherType, String what) {
        :
    }
    @Override public void checkType(int lineNum, Type correctType, String what) {
        :
    }
}
```

intType og doubleType opprettes som statiske variabler i modulen Types:

Types.java

```
package no.uio.ifi.cflat.types;

import no.uio.ifi.cflat.code.Code;
import no.uio.ifi.cflat.error.Error;
import no.uio.ifi.cflat.scanner.Token;
import static no.uio.ifi.cflat.scanner.Token.*;

public class Types {
    public static BasicType doubleType, intType;

    public static void init() {
        doubleType = new BasicType() {
            @Override public int size() {
                return 8;
            }

            @Override public String typeName() {
                return "double";
            }

            @Override public void genJumpIfZero(String jumpLabel) {
            }
        };
    }
};
```

Må alle klasser ha et navn?

Anonyme subklasser

Vanligvis deklarerer vi klasser og subklasser slik:

Bil.java

```
abstract class Bil {  
    abstract double antPassasjerer();  
}
```

Sportsbil.java

```
class Sportsbil extends Bil {  
    @Override double antPassasjerer() { return 1; }  
}
```

Bilregister.java

```
class Bilregister {  
    public static void main(String arg[]) {  
        Bil a = new Sportsbil();  
    }  
}
```

Må alle klasser ha et navn?

Hvis vi ikke trenger Sportsbil til noe annet, kan vi oppnå tilsvarende med en anonym subklasse:

Bil.java

```
abstract class Bil {  
    abstract double antPassasjerer();  
}
```

Bilregister2.java

```
class Bilregister2 {  
    public static void main(String arg[]) {  
        Bil a = new Bil() {  
            @Override double antPassasjerer() { return 1; }  
        };  
    }  
}
```

```
$ ls *.class
```

```
Bil.class
```

```
Bilregister2$1.class
```

```
Bilregister2.class
```



Må alle klasser ha et navn?

Vektortyper

Vektorer får en type som er en `ArrayType`:

```
package no.uio.ifi.cflat.types;

public class ArrayType extends Type {
    public int nElems;
    public Type elemType;

    public ArrayType(int n, Type t) {
        nElems = n; elemType = t;
    }

    @Override public int size() {
        return nElems*elemType.size();
    }

    @Override public String typeName() {
        return elemType.typeName() + " array";
    }

    @Override public void checkSameType(int lineNum, Type otherType, String what) {
    }

    @Override public void checkType(int lineNum, Type correctType, String what) {
```



Testutskrift

Det er lett å gjøre feil når man programmerer noe såpass komplisert som en kompilator. Det lureste er å godta dette og heller finne teknikker for å oppdage feilen.

- logP** avslører om man gjør riktige valg i jernbandediagrammene. La hver parse gi lyd fra seg.
- logT** sjekker om analysetreet ble riktig ved å skrive det ut etterpå.

Vårt testprogram

```
int pot2 (int x)
{
    int p2;    p2 = 1;
    while (2*p2 <= x) { p2 = 2*p2; }
    return p2;
}

int x;

int main ()
{
    int v;    v = getint();
    x = pot2(v);    putint(x);    putchar(10);
}
```


-logP

```

1: int pot2 (int x)
Parser: <program>
Parser: <func decl>
Parser: <param decl>
2: {
3:   int p2;   p2 = 1;
Parser: </param decl>
Parser: <func body>
Parser: <var decl>
Parser: </var decl>
Parser: <statm list>
Parser: <statement>
Parser: <assign-statm>
Parser: <assignment>
Parser: <variable>
Parser: </variable>
4:   while (2*p2 <= x) { p2 = 2*p2; }
Parser: <expression>
Parser: <term>
Parser: <factor>
Parser: <operand>
Parser: <number>
Parser: </number>
Parser: </operand>

```

```

Parser: </factor>
Parser: </term>
Parser: </expression>
Parser: </assignment>
Parser: </assign statm>
Parser: </statement>
Parser: <statement>
Parser: <while-statm>
Parser: <expression>
Parser: <term>
Parser: <factor>
Parser: <operand>
Parser: <number>
Parser: </number>
Parser: </operand>
Parser: <factor operator>
Parser: </factor operator>
Parser: <operand>
Parser: <variable>
Parser: </variable>
Parser: </operand>
Parser: </factor>
Parser: </term>
Parser: <rel operator>
Parser: </rel operator>
Parser: <term>

```

Implementasjon

Alle parse-metoder må kalle

```
Log.enterParser("<while-statm>");
```

(eller tilsvarende) ved oppstart og

```
Log.leaveParser("</while-statm>");
```

ved avslutning. Innrykk må håndteres automatisk.

Korrekt parsing av treet sjekkes enkelt ved å regenerere det (såkalt «pretty-printing»):

```
Tree:    int pot2 (int x)
Tree:    {
Tree:        int p2;
Tree:
Tree:        p2 = 1;
Tree:        while (2 * p2 <= x) {
Tree:            p2 = 2 * p2;
Tree:        }
Tree:        return p2;
Tree:    }
Tree:
Tree:    int x;
Tree:
Tree:    int main ()
Tree:    {
Tree:        int v;
Tree:
Tree:        v = getint();
Tree:        x = pot2(v);
Tree:        putint(x);
Tree:        putchar(10);
Tree:    }
```

Et eksempel

```
class WhileStatm extends Statement {
    Expression test = new Expression();
    StatmList body = new StatmList();

    @Override void parse() {
        Log.enterParser("<while-statm>");

        Scanner.readNext();
        Scanner.skip(leftParToken);
        test.parse();
        Scanner.skip(rightParToken);
        Scanner.skip(leftCur1Token);
        body.parse();
        Scanner.skip(rightCur1Token);

        Log.leaveParser("</while-statm>");
    }
}
```

Alle klasser som utvider SyntaxUnit, må ha en metode printTree:

```
@Override void printTree() {  
    Log.wTree("while ("); test.printTree(); Log.wTreeLn(") {");  
    Log.indentTree();  
    body.printTree();  
    Log.outdentTree();  
    Log.wTreeLn("}");  
}
```

I Log-modulen finnes noen nyttige metoder:

```
public static void wTree(String s) {
    if (curTreeLine.length() == 0) {
        for (int i = 1; i <= treeLevel; ++i) curTreeLine += " ";
    }
    curTreeLine += s;
}

public static void wTreeLn() {
    writeLogLine("Tree: " + curTreeLine);
    curTreeLine = "";
}

public static void wTreeLn(String s) {
    wTree(s); wTreeLn();
}

public static void indentTree() {
    //-- Must be changed in part 1:
}

public static void outdentTree() {
    //-- Must be changed in part 1:
}
```

Stilige programmer er bedre!

Programmeringsstil

Alle har sin programmeringsstil, men noen ganger kan det være lurt å følge en standard:

- Ved samarbeide
- Når man produserer kommersiell kode.

Sun har definert et forslag til standard: <http://www.oracle.com/technetwork/java/codeconventions-150003.pdf>.

Min mening er at

- Standardstilen er ofte bra men ikke i alle tilfeller.
- Alle bør prøve å utvikle sin personlige stil. En standardstil er et godt utgangspunkt.
- Det er lov å ha egne meninger om hva som er god stil.

Standard Java

```
class EasterJava {
    int easter(int y) {
        int g = y % 19 + 1;
        int c = y / 100 + 1;
        int x = 3 * c / 4 - 12;
        int z = (8 * c + 5) / 25 - 5;
        int d = 5 * y / 4 - x - 10;
        int e = (11 * g + 20 + z - x) % 30;
        int n;

        if (e == 25 && g > 11 || e == 24) {
            ++e;
        }
        n = 44 - e;
        if (n < 21) {
            n += 30;
        }
        n += 7 - (d + n) % 7;
        return n;
    }
}
```

Min stil

```
class Easter {
    int easter(int y) {
        int g = y%19 + 1;
        int c = y/100 + 1;
        int x = 3*c/4 - 12;
        int z = (8*c+5)/25 - 5;
        int d = 5*y/4 - x - 10;
        int e = (11*g + 20 + z - x)%30;
        if (e==25 && g>11 || e==24) ++e;

        int n = 44 - e;
        if (n < 21) n += 30;

        n += 7 - (d+n)%7;
        return n;
    }
}
```

Suns forslag

Klasser

Hver klasse bør ligge i sin egen kildefil.

Klasse-filer bør inneholde følgende (i denne rekkefølgen):

- 1 De aller viktigste opplysningene om filen:

```
/*  
 * Klassens navn  
 *  
 * Versjonsinformasjon  
 *  
 * Copyrightangivelse  
 */
```

- 2 Alle import-spesifikasjonene.
- 3 JavaDoc-kommentar for klassen.
- 4 Selve klassen.



Variable

- Variable bør deklarerer én og én på hver linje: *(Uenig!)*

```
int level;  
int size;
```

- De bør komme først i {}-blokken (dvs ikke etter noen setninger). *(Uenig!)*

- Lokale for-indekser er helt OK:

```
for (int i = 1; i <= 10; ++i) {  
    ...  
}
```

- Om man kan initialisere variablene samtidig med deklarasjonen, er det er fordel.

Setninger

- Enkle setninger bør stå én og én på hver linje: (*Uenig!*)

```
i = 1;  
j = 2;
```

- Sammensatte setninger:

```
do {  
  setninger;  
} while (uttrykk);  
  
if (uttrykk) {  
  setninger;  
}  
  
if (uttrykk) {  
  setninger;  
} else if (uttrykk) {  
  setninger;  
} else if (uttrykk) {  
  setninger;  
}
```

```
for (init; betingelse; oppdatering) {
    setninger;
}

return uttrykk;

while (uttrykk) {
    setninger;
}

try {
    setninger;
} catch (ExceptionClass e) {
    setninger;
}
```

```
switch (uttrykk) {  
  case xxx:  
    setninger;  
    break;  
  
  case xxx:  
    setninger;  
    break;  
  
  default:  
    setninger;  
    break;  
}
```

- Sammensatte setninger skal alltid ha {} rundt innmaten.
- Innmaten skal indenteres 4 posisjoner.

(Uenig!)

Navn

Navn bør velges slik:

Type navn	Kapitalisering	Hva slags ord	Eksempel
Klasser	XxxxXxxx	Substantiv som beskriver objektene	IfStatement
Metoder	xxxxXxxx	Verb som angir hva metoden gjør	readToken
Variable	xxxxXxxx	Korte substantiver; «bruk-og-kast-variable» kan være på én bokstav	curToken, i
Konstanter	XXXX_XX	Substantiv	MAX_MEMORY



Utseende

- Linjer skal ha maks 80 tegn. Lange linjer bør deles
 - etter et komma eller
 - før en operator (som + eller &&).
- Blanke linjer
Sett inn doble blanke linjer
 - mellom klasser.Sett inn enkle blanke linjer
 - mellom metoder,
 - mellom variabeldeklarasjonene og første setning i metoder eller
 - mellom ulike deler av en metode.

God dokumentasjon er viktig!

Dokumentasjon

God dokumentasjon er nødvendig, og mer nødvendig jo større programmene blir.

```
/* Sort array a with n elements using 'bubble sort'. */
```

```
void bubble(int a[], int n)
{
    int i, temp, n_swaps;

    do {
        n_swaps = 0;
        for (i=0; i<n-1; ++i)
            if (a[i]>a[i+1]) {
                temp = a[i]; a[i] = a[i+1]; a[i+1] = temp;
                ++n_swaps; }
    } while (n_swaps > 0);
}
```



God dokumentasjon er viktig!

Alle programmeringsspråk har muligheter til å legge inn kommentarlinjer. Det er ikke gitt at programmene blir mer lettleste av den grunn:

```
/* Sort array 'a' with 'n' elements using
   the technique called "bubble sort". */

void bubble(int a[], int n)
{
    /* Variables:
       i:      loop index
       temp:   temporary variable used during swapping
       n_swaps: count the number of swaps done */
    int i, temp, n_swaps;

    /* Loop until no swaps were performed, as this implies
       that the array is completely sorted. */
    do {
        /* Initialize the variables used in the loop. */
        n_swaps = 0;
        for (i=0; i<n-1; ++i)
            /* In a bubble sort, all adjacent array elements are
               compared. If they are out of order, they are
               swapped*/
            if (a[i]>a[i+1]) {
                /* Swapping uses a temporary variable. */
                temp = a[i]; a[i] = a[i+1]; a[i+1] = temp;
                /* Remember to update the swap counter. */
                ++n_swaps; }
    } while (n_swaps > 0);
    /* The array should now be completely sorted. */
}
```



Men det er flere måter å gjøre det på

Alternative løsninger

- man-filer i Unix
- eget dokumentasjonsdokument
- Perls **Pod**-format

Javas løsning

Java har en god løsning

The screenshot shows a Mozilla Firefox browser window displaying the Java Platform Standard Edition 7 API documentation for the `Integer` class. The browser's address bar shows the URL `http://download.oracle.com/javase/7/docs/api/`. The page title is "Integer (Java Platform SE 7)". The left sidebar shows a navigation menu with "Overview", "Package", "Use Tree", "Deprecated", "Index", and "Help". The main content area displays the class signature `java.lang.Integer` and its inheritance hierarchy: `java.lang.Object` and `java.lang.Number`. It lists implemented interfaces: `Serializable` and `Comparable<Integer>`. The class description states: "The Integer class wraps a value of the primitive type `int` in an object. An object of type `Integer` contains a single field whose type is `int`. In addition, this class provides several methods for converting an `int` to a `String` and a `String` to an `int`, as well as other constants and methods useful when dealing with `int`." It also includes a note about the implementation of the "bit twiddling" methods. Below the description, there are sections for "Field Summary" and "Constructor Summary".

Modifier and Type	Field and Description
static <code>int</code>	<code>MAX_VALUE</code> A constant holding the maximum value an <code>int</code> can have, 2 ³¹ -1.
static <code>int</code>	<code>MIN_VALUE</code> A constant holding the minimum value an <code>int</code> can have, -2 ³¹ .
static <code>int</code>	<code>SIZE</code> The number of bits used to represent an <code>int</code> value in two's complement binary form.
static <code>Class<Integer></code>	<code>TYPE</code> The class instance representing the primitive type <code>int</code> .

Hvordan skrive JavaDoc-kommentarer?

Kommentarene (i HTML-kode!) for klasser ser slik ut:

```
/**  
 * Én setning som kort beskriver klassen  
 * Mer forklaring  
  
 *      :  
 * @author navn  
 * @author navn  
 * @version dato  
 */
```

Kommentarer for metoder ser slik ut:

```
/**  
 * Én setning som kort beskriver metoden  
 * Ytterligere kommentarer  
  
 * ⋮  
 * @param navn1 Kort beskrivelse av parameteren  
 * @param navn2 Kort beskrivelse av parameteren  
 * @return Kort beskrivelse av returverdien  
 * @see navn3  
 */
```

Til sist kan programmet javadoc lage HTML-filene med dokumentasjonen.

«Lesbar programmering»

Donald Knuth var først ute med «literate programmering» og har inspirert mange lignende opplegg:

- 1 Programmet deles opp i passe små enheter som skrives i den rekkefølgen som er enklest å forklare.
- 2 Dokumentasjon skrives i samme filen som programkoden.
- 3 Programmet *weave* henter frem dokumentasjonskoden (i \LaTeX eller noe annet) for prosessering.
- 4 Programmet *tangle* henter frem kildekoden for kompilering.

Donald Knuths løsning

```
\documentclass[12pt,a4paper]{webzero}
\usepackage[latin1]{inputenc}
\usepackage[T1]{fontenc}
\usepackage{amssymb,mathpazo,textcomp}

\title{Bubble sort}
\author{Dag Langmyhr\\ Department of Informatics\\
  University of Oslo\\[5pt] \texttt{dag@ifi.uio.no}}

\begin{document}
\maketitle

\noindent This short article describes \emph{bubble
  sort}, which quite probably is the easiest sorting
  method to understand and implement.
  Although far from being the most efficient one, it is
  useful as an example when teaching sorting algorithms.

  Let us write a function \texttt{bubble} in C which sorts
  an array \texttt{a} with \texttt{n} elements. In other
  words, the array \texttt{a} should satisfy the following
  condition when \texttt{bubble} exits:
  \[
  \forall i, j \in \mathbb{N}: 0 \leq i < j < \mathbb{N}
  \Rightarrow \texttt{a}[i] \leq \texttt{a}[j]
  \]
```



```
<<bubble sort>>=  
void bubble(int a[], int n)  
{  
  <<local variables>>  
  
  <<use bubble sort>>  
}
```

@
Bubble sorting is done by making several passes through the array, each time letting the larger elements 'bubble' up. This is repeated until the array is completely sorted.

```
<<use bubble sort>>=  
do {  
  <<perform bubbling>>  
} while (<<not sorted>>);  
@
```

Donald Knuths løsning

Each pass through the array consists of looking at every pair of adjacent elements; \footnote{We could, on the average, double the execution speed of \texttt{bubble} by reducing the range of the \texttt{for}-loop by~1 each time.

Since a simple implementation is the main issue, however, this improvement was omitted.} if the two are in the wrong sorting order, they are swapped:

```
<<perform bubbling>>=  
<<initialize>>  
for (i=0; i<n-1; ++i)  
  if (a[i]>a[i+1]) { <<swap a[i] and a[i+1]>> }  
@
```

The \texttt{for}-loop needs an index variable \texttt{i}:

```
<<local var...>>=  
int i;  
@  
Swapping two array elements is done in the standard way using an auxiliary variable \texttt{temp}. We also increment a swap counter named \texttt{n\_swaps}.
```

```
<<swap ...>>=  
temp = a[i]; a[i] = a[i+1]; a[i+1] = temp;  
++n_swaps;  
@
```

Donald Knuths løsning

The variables `\texttt{temp}` and `\texttt{n_swaps}` must also be declared:

```
<<local var...>>=  
int temp, n_swaps;  
@
```

The variable `\texttt{n_swaps}` counts the number of swaps performed during one ‘‘bubbling’’ pass. It must be initialized prior to each pass.

```
<<initialize>>=  
n_swaps = 0;  
@
```

If no swaps were made during the ‘‘bubbling’’ pass, the array is sorted.

```
<<not sorted>>=  
n_swaps > 0  
@
```

```
\wzvarindex \wzmetaindex  
\end{document}
```

Donald Knuths løsning

Bubble sort

Dag Langmyhr
Department of Informatics
University of Oslo
dag@ifi.uio.no
August 8, 2008

This short article describes *bubble sort*, which quite probably is the easiest sorting method to understand and implement. Although far from being the most efficient one, it is useful as an example when teaching sorting algorithms.

Let us write a function `bubble` in C which sorts an array `a` with `n` elements. In other words, the array `a` should satisfy the following condition when bubble exits:

$$\forall i, j \in \mathbb{N} : 0 \leq i < j < n \Rightarrow a[i] \leq a[j]$$

```
#1 (bubble sort) =
{ void bubble(int a[], int n)
{ (local variables #1(p.1))
*
* (use bubble sort #1(p.1))
* }
(This code is not used.)
```

Bubble sorting is done by making several passes through the array, each time letting the larger elements “bubble” up. This is repeated until the array is completely sorted.

```
#2 (use bubble sort) =
* do {
* (perform bubbling #1(p.1))
* } while ((not sorted) #1(p.2));
(This code is used in #1 (p.1).)
```

Each pass through the array consists of looking at every pair of adjacent elements,¹ if the two are in the wrong sorting order, they are swapped:

```
#3 (perform bubbling) =
= (initialize #1(p.2))
= for (i=0; i<n-1; ++i)
= if (a[i]>a[i+1]) { (swap a[i] and a[i+1]) #1(p.2) }
(This code is used in #2 (p.1).)
```

The `for`-loop needs an index variable `i`:

```
#4 (local variables) =
= int i;
(This code is extended in #41(p.2). It is used in #1 (p.1).)
```

¹We could, on the average, double the execution speed of bubble by reducing the range of the `for`-loop by 1 each time. Since a simple implementation is the main issue, however, this improvement was omitted.

Swapping two array elements is done in the standard way using an auxiliary variable `temp`. We also increment a swap counter named `n_swaps`.

```
#5 (swap a[i] and a[i+1]) =
= temp = a[i]; a[i] = a[i+1]; a[i+1] = temp;
= ++n_swaps;
(This code is used in #3 (p.1).)
```

The variables `temp` and `n_swaps` must also be declared:

```
#41 (local variables #1(p.1)) =
= int temp, n_swaps;
```

The variable `n_swaps` counts the number of swaps performed during one “bubbling” pass. It must be initialized prior to each pass.

```
#6 (initialize) =
= n_swaps = 0;
(This code is used in #3 (p.1).)
```

If no swaps were made during the “bubbling” pass, the array is sorted.

```
#7 (not sorted) =
= n_swaps > 0
(This code is used in #2 (p.1).)
```



Råd nr 1: Forstå problemet!

Forstå hva du skal gjøre *før* du begynner å programmere.

- Skriv noen korte kodesnutter i C^b.
- Tegn syntakstrærne deres.
- Studér eksemplet med språket E i øvelsesoppgavene.

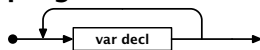
NB!

På dette stadium kan man samarbeide så mye man ønsker!

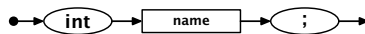
Råd nr 2: Start med noe enkelt!

Ingen bør forvente at de kan skrive all koden og så bare virker den. Start med et enkelt programmeringsspråk

program



var decl



og få det til å virke først. Utvid etter hvert. Sjekk hver utvidelse før du går videre.

Råd nr 3: Ikke sitt og stirr på koden!

Når programmet ikke virker:

- 1 Se på *siste versjon* av programkoden.
- 2 Siden du arbeider i små steg er feilen sannsynligvis i de siste linjene du endret.
- 3 Hvis du ikke har funnet feilen i løpet av fem minutter, gå over til *aktiv feilsøking*.

Råd nr 4: Les testutskriftene!

P-utskriftene forteller hvilke parse-metoder som kalles.

Anta at vi har programmet

```
int pot2 (int x)
{
  int p2;   p2 = 1;
}
:
:
```

Programmet gir en feilmelding i linje 1:

```
Expected a leftCurlyToken but found a leftParToken!
```

Hva er galt?

Svaret kan vi finne i **P**-utskriften:

```
1: int pot2 (int x)
Parser: <program>
Parser:   <func decl>
2: {
3:   int p2;   p2 = 1;
Parser:     <func body>
```

Utskriften skulle ha startet

```
1: int pot2 (int x)
Parser: <program>
Parser:   <func decl>
Parser:     <param decl>
2: {
3:   int p2;   p2 = 1;
Parser:     </param decl>
Parser:     <func body>
```

T-utskriften viser en utskrift av det genererte treet. Anta at vi har det samme testprogrammet

```
int pot2 (int x)
{
  int p2;   p2 = 1;
}
:
:
```

Hvis **T**-utskriften er

```
int pot2 (int x)
{
  p2 = 1;
}
:
:
```

så vet vi at lokale deklarasjoner i funksjoner ikke settes opp riktig (eller at det er feil i **T**-utskriften ☺).

Råd nr 5: Lag egne testutskrifter

Her er litt (muligens feilaktig) kode fra StatmList.parse:

```
class StatmList extends SyntaxUnit {
    Statement firstStatement = null;

    @Override void parse() {
        Log.enterParser("<statm list>");

        Statement lastStatm = null;
        while (Scanner.curToken != rightCurlyToken) {
            Log.enterParser("<statement>");
            Statement s = ...;
            s.parse();

            if (lastStatm != null) firstStatm = lastStatm = s;
            else lastStatm.nextStatm = lastStatm = s;
            Log.leaveParser("</statement>");
        }
    }
}
```



Anta at vi får melding om null-peker i linjen med
`lastStatm.nextStatm = lastStatm = sx;`. Slikt skal ikke skje.

Mitt råd er å legge inn noe à la

```
while (...) {  
    System.out.println("StatmList.parse: " +  
        "lastStatm er " +  
        (lastStatm==null ? "null" : "ikke null"));
```

Slik skal koden være:

```
class StatmList extends SyntaxUnit {
    Statement firstStatement = null;

    @Override void parse() {
        Log.enterParser("<statm list>");

        Statement lastStatm = null;
        while (Scanner.curToken != rightCurlyToken) {
            Log.enterParser("<statement>");
            Statement s = ...;
            s.parse();

            if (lastStatm == null) firstStatm = lastStatm = s;
            else lastStatm.nextStatm = lastStatm = s;
            Log.leaveParser("</statement>");
        }
    }
}
```

Råd nr 6: Behold testutskriftene!

Når feilen er funnet, bør man la testutskriften forbli i koden. Man kan få bruk for den igjen.

Derimot bør man kunne slå den av eller på:

- Det mest avanserte er å bruke opsjoner på kommandolinjen:

```
java Cflat -debugSL.a testprog.cflat
```

- Det fungerer også godt å benytte statusvariable:

```
static boolean debugSL_a = true;
    ⋮
if (debugSS_a) {
    System.out.println("...");
}
```

Råd nr 7: Mistro din egen kode!

Det er altfor lett å stole på at ens egen kode andre steder er korrekt.

Løsningen er å legge inn *assertions* som bare sjekker at alt er som det skal være.

```
class WhileStatm extends Statement {
    Expression test = new Expression();
    StatmList body = new StatmList();

    @Override void parse() {
        Log.enterParser("<while-statm>");

        assert Scanner.curToken==whileToken:
            "While-setning starter ikke med 'while'!";
        Scanner.readNext();
        Scanner.skip(leftParToken);
        test.parse();
        Scanner.skip(rightParToken);
        Scanner.skip(leftCurToken);
        body.parse();
        Scanner.skip(rightCurToken);

        Log.leaveParser("</while-statm>");
    }
}
```



NB!

Husk å kjøre med
 java -ea Cflat
for å slå på mekanismen.

Råd nr 8: Sjekk spesielt på `Scanner.readNext()`!

Det er lett å kalle `readNext()` for ofte eller for sjelden.
Her er reglene som parse-metodene må følge:

- 1 Når man kaller `parse`, skal første symbol være lest inn.
- 2 Når man returnerer fra en `parse`, skal første symbol *etter* konstruksjonen være lest.

Vær spesielt oppmerksom der du har forgreninger og løkker i jernbaniediagrammet.

Råd nr 9: Ta kopier daglig eller oftere!

Programmering er mye prøving og feiling. Noen ganger må man bare glemme alt man gjorde den siste timen.

Det finnes systemer for versjonskontroll som man bør lære seg før eller siden. I mellomtiden kan man ha nytte av

- 1 Ta en kopi av Java-filen hver gang du starter med å legge inn ny kode.
- 2 Ta uansett en kopi hver dag (om noe som helst er endret).

Råd nr 10: Fordel arbeidet!

Dere er to om jobben. Selv om begge må kjenne til hovedstrukturen, kan man fordele programmeringen.

Forslag

- 1 Én tar deklarasjoner.
- 2 Én tar setninger og uttrykk.

Men ...

- Snakk ofte sammen.
- Planlegg hvordan dere bruker filene så ikke den ene ødelegger det den andre har gjort.

Råd nr 11: Bruk hjelpemidlene

Spør gruppelærerne!

De er tilgjengelige under gruppetimene og svarer på e-post til andre tider.

Orakel

De siste ukene før oblig-fristene vil gruppelærerne bare jobbe med å svare på spørsmål.

Les kompendiet

Stoffet er forklart med flere detaljer enn det er mulig på forelesningene.

Råd nr 12: Start *nå!*

Det tar typisk 20-100 timer å programmere Del-1.

Påtrengende spørsmål

Det er 20 arbeidsdager til 25. oktober. Hvor mange timer per dag blir det?