Dagens tema: Resten av det dere trenger til del 1

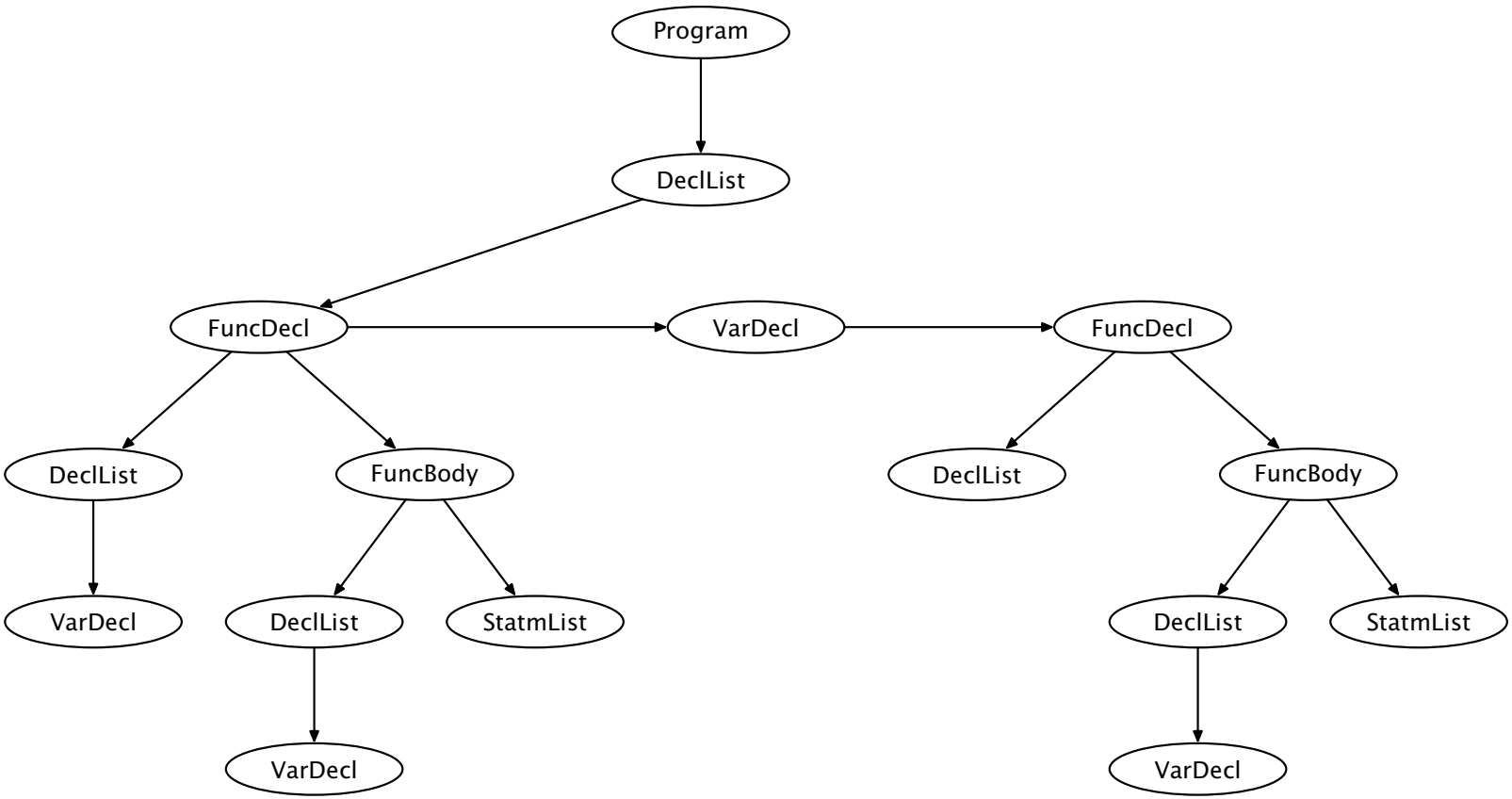
- Hvordan sjekke navn?
- Testutskrifter
- Programmeringsstil
- 12 gode råd

Prosjektet

Utifra dette RusC-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 i grammatikken, men det er lov å avvike fra dette.

- Nye klasser **declaration** og **decl list** er nyttig.
- Vi kan slå sammen **func decl** og **func body** til én klasse om vi ønsker det.

Sjekking av navn

En kompilator må også sjekke riktig navnebruk:

- Det må ikke forekomme dobbeltdeklarasjoner.
- Alle navn må være deklarert.
- Navnet må brukes riktig (enkel variabel kontra array kontra funksjon).

Lagring av navn

Enkeltnavnene bør lagres i deklarasjonsobjektet:

```
abstract class Declaration extends SyntaxUnit {
    protected String name = null;
    protected Declaration nextDecl = null;

    public abstract void checkWhetherArray();
    public abstract void checkWhetherFunction(int nParamsUsed);
    public abstract void checkWhetherSimpleVar();
}

class FuncDecl extends Declaration {
    ...

    public static FuncDecl parse(...) {...}

    public void checkWhetherArray() {...}
    public void checkWhetherFunction(int nParamsUsed) {...}
    public void checkWhetherSimpleVar() {
        Scanner.illegal(name+" is a function and no variable!");
    }
}
```

For å sjekke at et navn er riktig brukt, kalles en CheckWhether. . . -metode.

Eksempel

Vi har funnet setningen f() som er et ⟨function call⟩. Navnet f må da være deklarert som en funksjon med 0 parametre. Vi sjekker dette slik:

1. Finn Declaration fDecl; . . . fDecl = . . . ;
2. Kall **fDecl.checkWhetherFunction(0);**

   Om f er galt deklarert, skrives det ut en feilmelding og kompileringen stopper.

Det er nyttig med en egen klasse for en deklarasjonsliste:

```
class DeclList extends SyntaxUnit {
    Declaration firstDecl;
    DeclList outerScope;

    DeclList (DeclList outer) {...}

    void addDecl(Declaration d) {...}

    Declaration findDecl(String n) {...}
}
```

Hvordan finne et navn?

I RusC-programmet kan vi ha følgende setning:

```
x = a[4] + 1;
```

Hvilken x er det snakk om, og hvilken a?

Lokale variable

Det enkleste er å sjekke om variabelen er en lokal variabel:

```
int f ()
{
   int c;   c = getchar() + 32;
   putchar(c);
}
```

La parse få med en parameter som peker til den lokale deklarasjonslisten:

```
public static WhileStatm parse(DeclList decls) {
    :
}
```

Globale navn

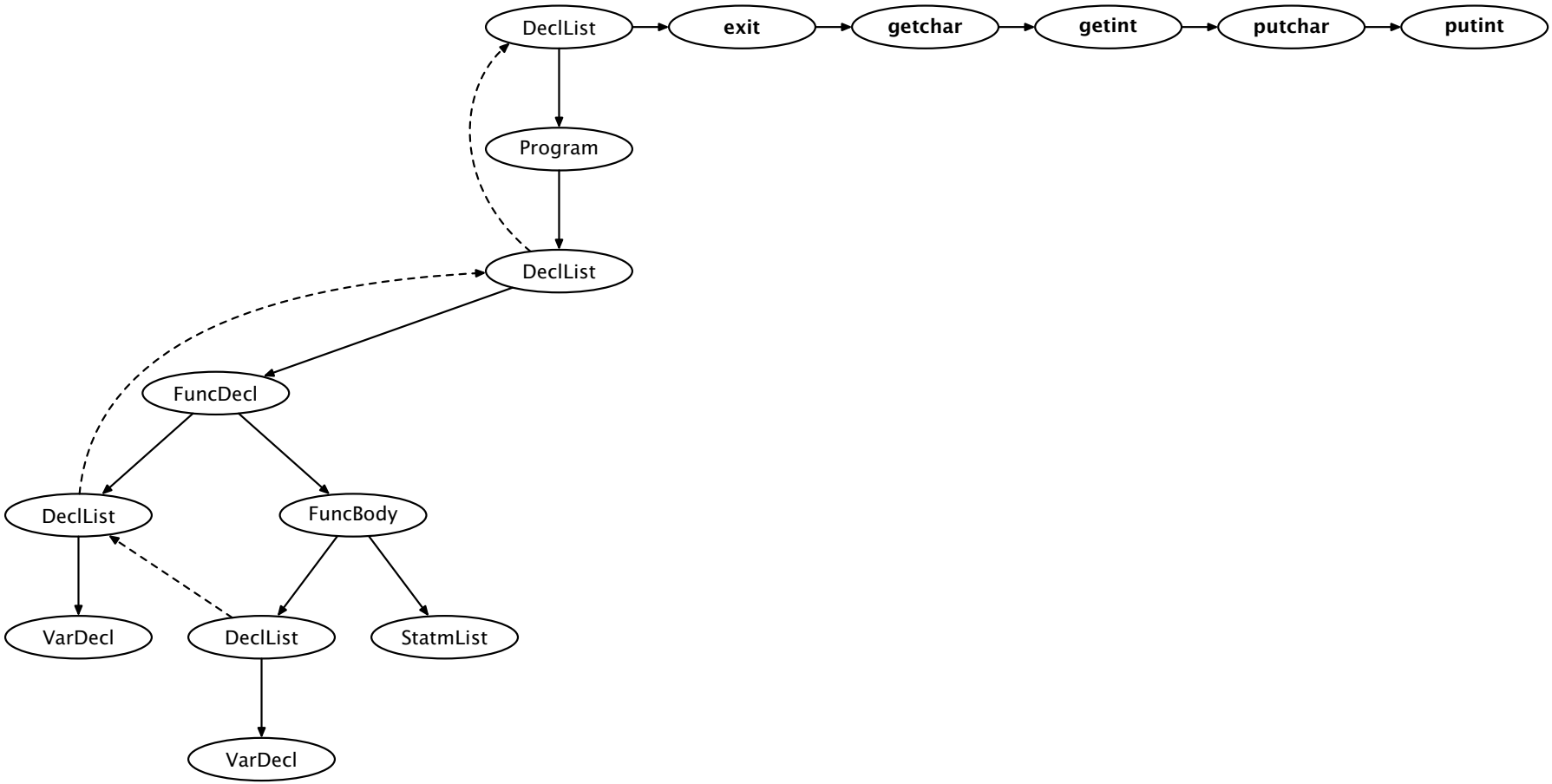
Navn kan imidlertid være deklarert «lenger ute». I Rusc kan vi ha opptil fire nivåer:

```
int a;

int f (int b)
{
   int c;

   ... c ... b ... a ... getint() ...
```

Hvordan finner globale navn?

Det enkleste er å la hvert DeclList-objekt ha en peker til sin nærmeste omliggende deklarasjonsliste.

Hvordan finne navn?



Skjuling av navn

Som i de fleste språk vil en indre deklarasjon skjule en ytre.

```
int a;

int f (int a)
{
   int a;

   ... a ...
}
```

Dette håndteres automatisk av vårt opplegg.

Deklarasjonsrekkefølgen

Husk at navn i RusC først er kjent *etter* at de er deklarert.

```
int f1 ()
{
   outint(v);
}

int v;
    :
```

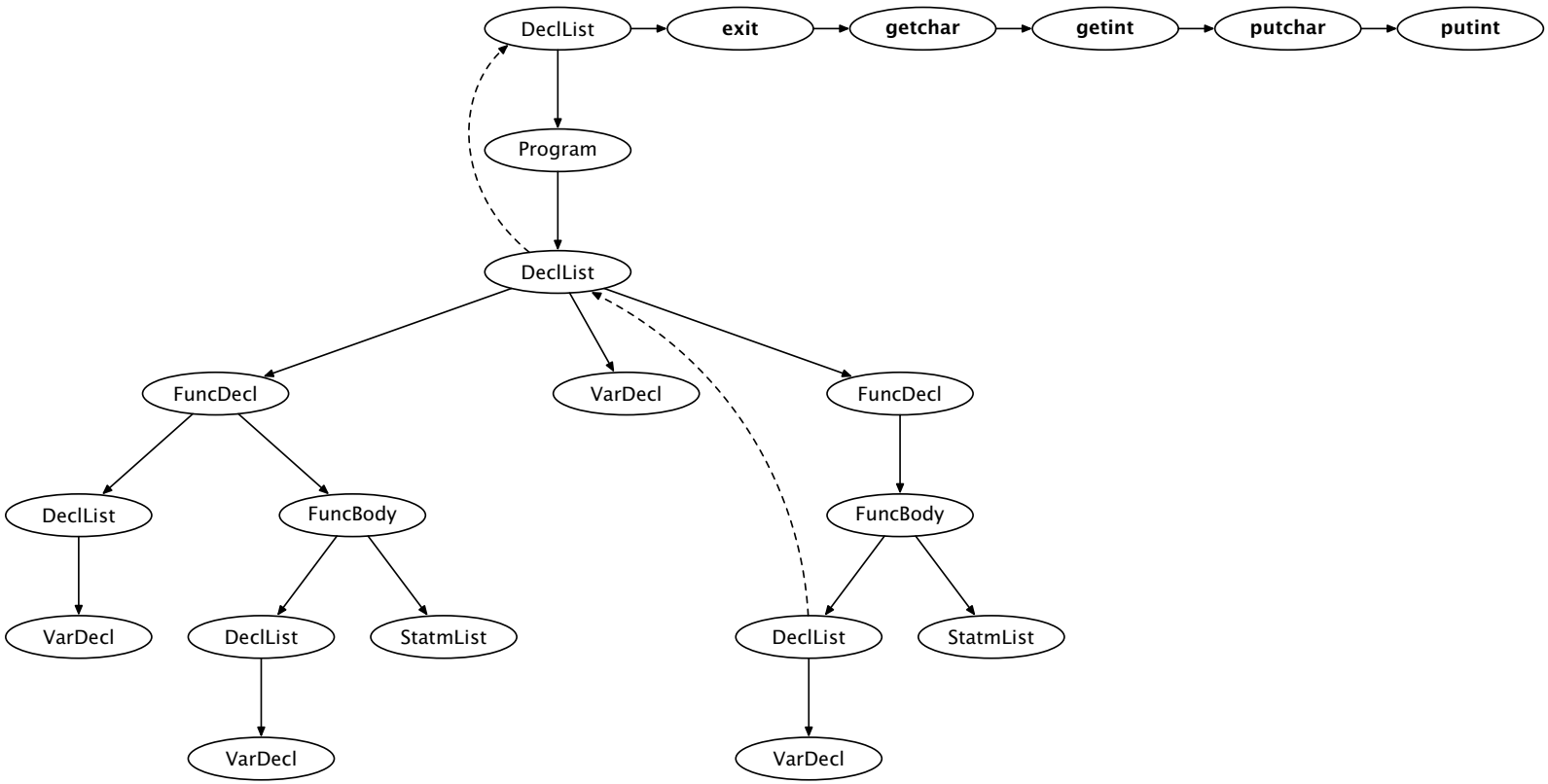
. . . så dette programmet skal gi feilmelding.

Noen ganger er dette ganske viktig:

```
int f1 (int a)
{
   putint(a);
}

int putint (int a)
{
   putchar(a);
}

int f2 (int a)
{
   putint(a);
}
```

Dette håndteres også automatisk når vi gjør det samtidig med parseringen.

Hvordan finne navn?



Testutskrifter

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 jernbanediagrammene. 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:                   <number>
Parser:                   </number>
Parser:                 </expression>
Parser:               </assignment>
Parser:             </assign statm>
Parser:           </statement>
Parser:           <statement>
Parser:             <while-statm>
Parser:               <expression>
Parser:                 <number>
Parser:                 </number>
Parser:                 <operator>
Parser:                 </operator>
Parser:                 <variable>
Parser:                 </variable>
Parser:                 <operator>
Parser:                 </operator>
Parser:                 <variable>
Parser:                 </variable>
Parser:               </expression>
Parser:               <statm list>
Parser:                 <statement>
Parser:                   <assign-statm>
Parser:                     <assignment>
Parser:                       <variable>
Parser:                       </variable>
Parser:                       <expression>
Parser:                         <number>
Parser:                         </number>
Parser:                         <operator>
Parser:                         </operator>
Parser:                         <variable>
```

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 parsering 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;
    StatmList body;

    public static WhileStatm parse(DeclList decls) {
        Log.enterParser("<while-statm>");

        WhileStatm w = new WhileStatm();
        Scanner.readNext();
        Scanner.skip(Token.leftParToken);
        w.test = Expression.parse(decls);
        Scanner.skip(Token.rightParToken);
        Scanner.skip(Token.leftCurlToken);
        w.body = StatmList.parse(decls);
        Scanner.skip(Token.rightCurlToken);

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

```
public 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:
}
```

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://java.sun.com/docs/codeconv/CodeConventions.pdf.

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 deklareres 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 (uttykk) {
  setninger;
} else if (uttrykk) {
  setninger;
} else if (uttrykk) {
  setninger;
}
```

Suns forslag

```
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 har {} rundt innmaten. *(Uenig!)*
- Innmaten skal indenteres 4 posisjoner.

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 maks 80 tegn. For 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.

Råd nr 1: Start *nå*!

Det tar typisk 20–100 timer å programmere Del-1 om man ikke har gjort slikt før.

Påtrengende spørsmål

Det er 30 arbeidsdager til 27. oktober. Hvor mange timer per dag blir det?

Råd nr 2: Forstå problemet!

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

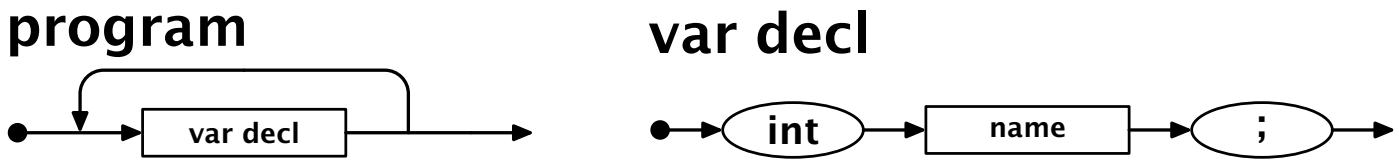
- Skriv noen korte kodesnutter i RusC.
- Tegn syntakstrærne deres.
- Studér eksemplet med språket E i øvelsesoppgavene.

NB!

På dette stadium kan man samarbeide så mye man ønsker!

Råd nr 3: Start med noe enkelt!

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



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

Råd nr 4: 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 5: 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 leftCurlToken but found a leftParToken!
```

Hva er galt?

Husk testutskriftene!

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:      <func params>
   2: {
   3:    int p2;   p2 = 1;
Parser:      </func params>
Parser:      <func body>
```

Husk testutskriftene!

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 6: Lag egne testutskrifter

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

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

  public static StatmList parse(DeclList decls) {
    Log.enterParser("<statm list>");

    StatmList sl = new StatmList();
    Statement lastStatement = null;
    Scanner.skip(Token.leftCurlToken);
    while (Scanner.curToken != Token.rightCurlToken) {
      Statement sx = Statement.parse(decls);
      if (lastStatement != null) sl.firstStatement = sx;
      else lastStatement.nextStatement = sx;
      lastStatement = sx;
    }
    Scanner.readNext();

    Log.leaveParser("</statm list>");
    return sl;
  }
```

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

Mitt råd er å legge inn noe à la

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

Slik skal koden være:

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

    public static StatmList parse(DeclList decls) {
        Log.enterParser("<statm list>");

        StatmList sl = new StatmList();
        Statement lastStatement = null;
        while (Scanner.curToken != Token.rightCurlToken) {
            Statement sx = Statement.parse(decls);
            if (lastStatement == null) sl.firstStatement = sx;
            else lastStatement.nextStatement = sx;
            lastStatement = sx;
        }

        Log.leaveParser("</statm list>");
        return sl;
    }
}
```

Råd nr 7: 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 Rusc -debugSS.a testprog.rusc
```

- Det fungerer også godt å benytte statusvariable:

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

Råd nr 8: 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.

Stol ikke på det du har skrevet!

```
class WhileStatm extends Statement {
  Expression test;
  StatmList body;

  public static WhileStatm parse(DeclList decls) {
    Log.enterParser("<while-statm>");

    WhileStatm w = new WhileStatm();

    assert Scanner.curToken==Token.whileToken:
      "While-setning starter ikke med 'while'!";
    Scanner.readNext();
    Scanner.skip(Token.leftParToken);
    w.test = Expression.parse(decls);
    Scanner.skip(Token.rightParToken);
    Scanner.skip(Token.leftCurlToken);
    w.body = StatmList.parse(decls,true);
    Scanner.skip(Token.rightCurlToken);

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

NB!

Husk å kjøre med

```
java -ea Rusc
```

for å slå på mekanismen.

Råd nr 9: 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 på if-tester og løkker.

Råd nr 10: 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 11: Fordel arbeidet!

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

Forslag

1. Én tar det som har med deklarasjoner.
2. Én tar det som har med setninger.

Men . . .

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

Råd nr 12: 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.

Vi har snakket om

- Navn lagres i deklarasjonsobjektet.
- parse får med peker til lokale deklarasjonsliste.
- Hver deklarasjonsliste peker på listen utenfor.
- Testutskriftene er svært nyttige.
- God programmeringsstil er også en hjelp.
- 12 gode råd.