Dagens tema:

- Kompilatorens struktur
  - Oppbyggingen
  - Pakker i Java
  - Enum-klasser i Java
- De ulike modulene
- Prosjektet
  - Hva skal **del-0** gjøre?
  - Testutskrifter
  - Siste råd og påbud

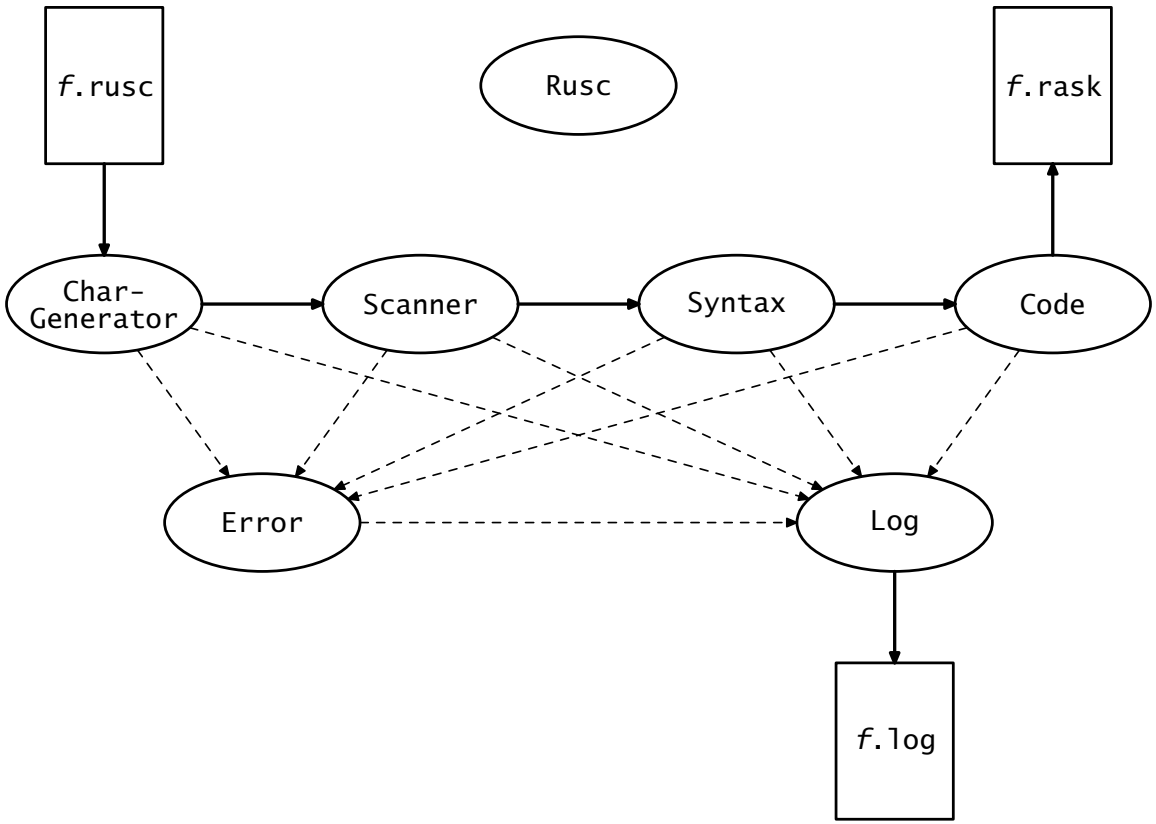
Prosjektet

Hvordan skriver man et større program som en kompilator?

Struktur

En fornuftig måte å dele opp problemet på er å se på hvor data flyter. Da er det enklere å kapsle inn deler av programmet.

Vår struktur



Noen programmeringsspråk har mekanismer for store moduler – men langt fra alle. Java har package.

Begrunnelse

Anta at vi skal utvikle et CAD-system. Et firma i India har laget en god GUI-modul.
Men, begge har en klasse Point.

Moduler kan løse dette problemet.

Alle filene som skal inngå i en Java-pakke starter med «package *navn*».

Pakkenavn bør bestå av opphavsstedets internettadresse (baklengs) og et lokalt navn. Vårt kompilatorprosjekt heter

no.uio.ifi.rusc

Eksempel

Filen P1/A.java:

```
package P1;

public class A {
    public static int x = 1;
}
```

(Under kompileringen må klassen ligge i et fil-tre som tilsvarer leddene i pakkenavnet; våre filer ligger i

no/uio/ifi/rusc/scanner/Scanner.java

og tilsvarende.)

Vi kan hente klasser fra alle pakker så lenge de finnes i CLASSPATH:

```
class B {
    public static void main (String arg[]) {
        System.out.println("P1.A.x = " + P1.A.x);
    }
}
```

Ifis standard CLASSPATH er:

```
$ printenv CLASSPATH
/local/opt/java/classes:/local/opt/java/classes/postgresql-8.3-603.jdbc4.jar:
/hom/dag/java/classes:.
```

Beskyttelse

Klasser kan beskyttes:

public	kan brukes fra andre pakker.
—	er usynlig utenfor pakken.

For klasseelementer gjelder:

private	er bare tilgjengelige i klassen.
protected	er for klassen og subklasser.
—	er bare for bruk innen pakken.
public	kan benyttes overalt.

For å unngå å skrive mange lange navn, kan vi importere klasser fra pakker:

```
import P1.A;

class B {
    public static void main (String arg[]) {
        System.out.println("P1.A.x = " + A.x);
    }
}
```

Vi kan også importere alle klassene fra en pakke:

```
import P1.*;

class B {
    public static void main (String arg[]) {
        System.out.println("P1.A.x = " + A.x);
    }
}
```

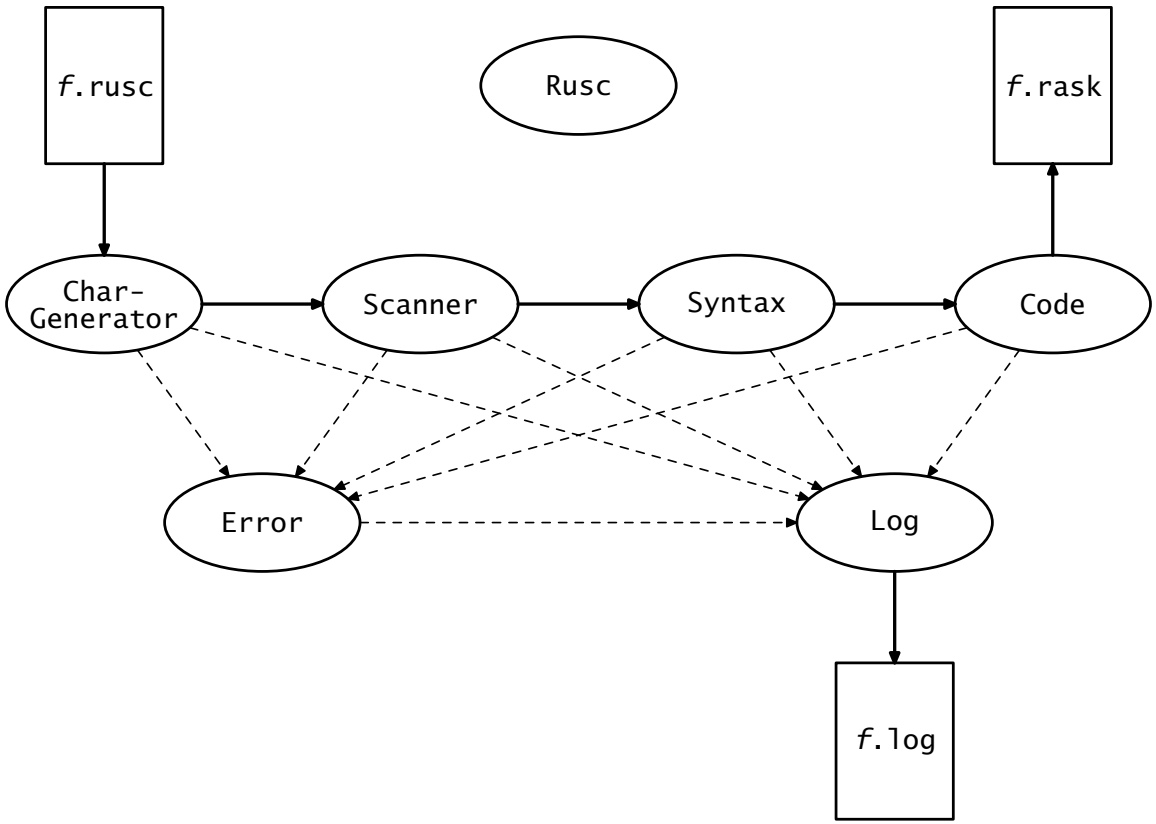
Hva kan en pakke inneholde?

En Java-pakke kan bare inneholde klasser. Vi trenger også data og metoder og vedtar derfor:

- I alle pakker finnes en klasse med samme navn som pakken (men stor forbokstav); den inneholder data og metoder vi trenger.
- Alle disse klassene har disse to metodene:

  `init` benyttes til initiering av pakken.
  `finish` avslutter pakken.

Vår struktur



Hovedprogrammet

```
package no.uio.ifi.rusc.rusc;


import java.io.*;
import no.uio.ifi.rusc.chargenerator.CharGenerator;
import no.uio.ifi.rusc.code.Code;
import no.uio.ifi.rusc.error.Error;
import no.uio.ifi.rusc.log.Log;
import no.uio.ifi.rusc.scanner.Scanner;
import no.uio.ifi.rusc.scanner.Token;
import no.uio.ifi.rusc.syntax.Syntax;
```

```
public class Rusc {
    public static final String version = "2009-08-17";

    public static String sourceName = null;

    /**
     * The actual "main program" of the RusC compiler.
     * It will initialize the various modules and start the
     * compilation (or module testing, if requested); finally,
     * it will terminate the modules.
     *
     * @param args The command line arguments.
     */
    public static void main(String[] args) {
        boolean testParser = false, testScanner = false;

        for (int opt_no = 0;  opt_no < args.length;  ++opt_no) {
            String opt = args[opt_no];
```

```
        if (opt.equals("-logC")) {
            Log.doLogCode = true;
        } else if (opt.equals("-logP")) {
            Log.doLogParser = true;
        } else if (opt.equals("-logT")) {
            Log.doLogTree = true;
        } else if (opt.equals("-logS")) {
            Log.doLogScanner = true;
        } else if (opt.equals("-testparser")) {
            testParser = true;
            Log.doLogParser = Log.doLogTree = true;
        } else if (opt.equals("-testscanner")) {
            testScanner = true;
            Log.doLogScanner = true;
        } else if (opt.equals("-version")) {
            System.err.println("This is Rusc (version "+version+")");
        } else if (opt.startsWith("-")) {
            Error.giveUsage();
        } else {
            if (sourceName != null) Error.giveUsage();
            sourceName = opt;
        }
    }
    if (sourceName == null) Error.giveUsage();
```

Modulen «Rusc»

```
        Error.init();  Log.init();  Code.init();
        CharGenerator.init();  Scanner.init();  Syntax.init();

        if (testScanner) {
            while (Scanner.nextToken != Token.eofToken) Scanner.readNext();
        } else {
            Syntax.parseProgram();
            if (Log.doLogTree) Syntax.printProgram();
            if (! testParser) Syntax.genCode();
        }

        Syntax.finish();  Scanner.finish();  CharGenerator.finish();
        Code.finish();  Log.finish();  Error.finish();
    }
}
```

Skanner

En kompilator *kan* lese og tolke en program tegn for tegn, men det er mye lettere om det kan gjøres *symbol for symbol*. Dette ordner en **skanner**.

CharGenerator leser programkoden linje for linje, fjerner #-kommentarlinjer og deler de andre linjene opp i enkelt-tegn.

Scanner setter tegnene sammen til symboler (og fjerner /*...*/-kommentarer).

```
/* Program som leser et tall v
   og skriver ut v+1. */

# Hovedprogrammet:
int main ()
{
   int v;

   /* Les data: */
   v = getint() + 1;
   /* Skriv svaret: */
   putint(v);   putchar(10);
}
```

har disse symbolene:

int	main	(	)	{	int	v	;	v	=					
getint	(	)	+	1	;	putint	(	v	)	;				
putchar	(	10	)	;	}									

Enum-klasser

Noen ganger har man diskrete data som kun kan ha et begrenset antall fast definerte verdier:

Kortfarge	Kløver, ruter, hjerter, spar
Tippetegn	Hjemmeseier, uavgjort, borteseier
Ukedag	Mandag, tirsdag, onsdag, torsdag, fredag, lørdag, søndag

Å representere disse med heltall er en halvgod løsning.

Java tilbyr **enum-klasser**:

```
enum Tippetegn {
    Hjemmeseier, Uavgjort, Borteseier;
}
```

Man oppgir alle verdiene (og kan legge inn ekstra metoder og data).

Dette er *nesten* det samme som

```
class Ttegn {
    private String name;

    private Ttegn(String s) {
        name = s;
    }

    public static final Ttegn
        Hjemmeseier = new Ttegn("Hjemmeseier"),
        Uavgjort = new Ttegn("Uavgjort"),
        Borteseier = new Ttegn("Borteseier");

    public String toString () {
        return name;
    }
}
```

Slik brukes denne klassen:

```
class Tipping {
    public static void main (String arg[]) {
        Tippetegn rekke[] = new Tippetegn[12+1];

        rekke[1] = Tippetegn.Hjemmeseier;
        rekke[2] = Tippetegn.Borteseier;
        rekke[3] = Tippetegn.Borteseier;

        for (int i = 1;  i <= 3;  ++i)
            System.out.print(rekke[i]+" ");
        System.out.println();
    }
}
```

```
> java Tipping
Hjemmeseier Borteseier Borteseier
```

Hva kan vi gjøre med enum-klasser?

- Tilordne verdier («rekke[i] = Tippetegn.Uavgjort»)
- Sjekke på likhet og ulikhet («rekke[1] == Tippetegn.Borteseier»)
- Skrive ut objektet («System.out.println(rekke[1])» som er det samme som «System.out.println(rekke[1].toString())»)

I vår skanner

Vår skanner kan levere følgende **token**:

```
package no.uio.ifi.rusc.scanner;

/*
 * class Token
 */

/*
 * The different kinds of tokens read by Scanner.
 */
public enum Token { addToken, assignToken, commaToken, divideToken, elseToken,
    eofToken, equalToken, forToken, greaterEqualToken, greaterToken,
    ifToken, intToken, leftBracketToken, leftCurlToken, leftParToken,
    lessEqualToken, lessToken, multiplyToken, nameToken, notEqualToken,
    noToken, numberToken, rightBracketToken, rightCurlToken, rightParToken,
    returnToken, semicolonToken, subtractToken, whileToken;

    public static boolean isOperator(Token t) {
        //-- Må endres i del 0:
        return false;
    }
}
```

Modulen scanner

Symbolene leses inn i curToken, nextToken og nextNextToken (samt i curName, curNumber, nextName, nextNumber, nextNextName og nextNextNumber).

```
package no.uio.ifi.rusc.scanner;


import no.uio.ifi.rusc.chargenerator.CharGenerator;
import no.uio.ifi.rusc.error.Error;
import no.uio.ifi.rusc.log.Log;


public class Scanner {
    public static Token curToken, nextToken, nextNextToken;
    public static String curName, nextName, nextNextName;
    public static long curNum, nextNum, nextNextNum;

    public static void init() {
        //-- Må endres i del 0:
    }

    public static void finish() {
        //-- Må endres i del 0:
    }
```

Lesingen skjer med readNext-metoden:

```
public static void readNext() {
    curToken = nextToken;  nextToken = nextNextToken;
    curName = nextName;  nextName = nextNextName;
    curNum = nextNum;  nextNum = nextNextNum;

    nextNextToken = Token.noToken;
    while (nextNextToken == Token.noToken) {
        //-- Må endres i del 0:
    }
    Log.noteToken(nextNextToken);
}
```

En nyttig hjelperutine:

```
private static boolean isLetterAZ(char c) {
    //-- Må endres i del 0:
    return false;
}
```

Modulen chargenerator

```
package no.uio.ifi.rusc.chargenerator;


import java.io.*;
import no.uio.ifi.rusc.error.Error;
import no.uio.ifi.rusc.log.Log;
import no.uio.ifi.rusc.rusc.Rusc;

public class CharGenerator {
    public static char curC, nextC;

    private static LineNumberReader sourceFile = null;
    private static String sourceLine;
    private static int sourcePos;
```

```
public static void init() {
    try {
        sourceFile = new LineNumberReader(new FileReader(Rusc.sourceName));
    } catch (FileNotFoundException e) {
        Error.error("Cannot read '" + Rusc.sourceName + "'!");
    }
    sourceLine = "";  sourcePos = 0;  curC = nextC = ' ';
    readNext();  readNext();
}

public static void finish() {
    if (sourceFile != null) {
        try {
            sourceFile.close();
        } catch (IOException e) {
            Error.error("Could not close source file!");
        }
    }
}
```

Metoden readNext leser neste tegn:

```
public static void readNext() {
    curC = nextC;
    if (! isMoreToRead()) return;

    //-- Må endres i del 0:
}
```

To nyttige metoder:

```
public static boolean isMoreToRead() {
    //-- Må endres i del 0:
    return false;
}

public static int curLineNum() {
    return (sourceFile == null ? 0 : sourceFile.getLineNumber());
}
```

Hva er en god feilmelding?

Ubrukelig

```
ERROR: Syntax error detected!
```

Noe bedre

```
ERROR: Syntax error found in line 217.
```

Enda bedre

```
ERROR: Syntax error found in line 217:
    x = x+1 } ;
```

Melding med mening

Meldingen bør fortelle hva som er galt:

```
ERROR in line 217: Semicolon expected.
    x = x+1 } ;
```

Hvor på linjen?

Meldingen bør fortelle hvor på linjen feilen er:

```
ERROR in line 217: Semicolon expected.
    x = x+1 } ;
**************^
```

Det er ikke alltid like lett!

Den beste meldingen

Meldingen bør angi hvorledes kompilatoren «tenker»:

```
ERROR in line 217:
Expected ';' at end of sentence but found '}'.
    x = x+1 } ;
```

Feil

Hva gjør man med feil?

- Før prøvde man å finne så mange feil som mulig.
- Vi skal stoppe med melding ved første feil.

Modulen error

```
package no.uio.ifi.rusc.error;


import no.uio.ifi.rusc.log.Log;

/*
 * Print error messages.
 */
public class Error {
    public static void error(String where, String message) {
        //-- Må endres i del 0:

        System.exit(1);

    }
}
```

Modulen «Error»

```
public static void error(String message) {
    error("", message);
}

public static void error(int lineNum, String message) {
    error("in line " + lineNum, message);
}

public static void giveUsage() {
    System.err.println("Usage: rusc [-log{C|P|S|T}] [-test{scanner|parser}] " +
                       "[-version] file");
    System.exit(2);
}
```

Siden de fleste feilene er relatert til lesingen av RusC-koden, er det nyttig med en egen metode i Scanner-modulen:

```
public static void illegal(String message) {
    Error.error(CharGenerator.curLineNum(), message);
}

public static void expected(String exp) {
    illegal(exp + " expected, but found a " + curToken + "!");;
}

public static void check(Token t) {
    if (curToken != t)
        expected("A " + t);
}
```

Modulen log

Brukeren kan slå av og på logging (med opsjoner som håndteres av Rusc-modulen).

```
package no.uio.ifi.rusc.log;


import java.io.*;
import no.uio.ifi.rusc.error.Error;
import no.uio.ifi.rusc.rusc.Rusc;
import no.uio.ifi.rusc.scanner.Scanner;
import no.uio.ifi.rusc.scanner.Token;

public class Log {
    public static boolean doLogCode = false, doLogParser = false,
        doLogScanner = false, doLogTree = false;

    private static String logName, curTreeLine = "";
    private static int nLogLines = 0, parseLevel = 0, treeLevel = 0;

    public static void init() {
        logName = Rusc.sourceName;
        if (logName.endsWith(".rusc"))
            logName = logName.substring(0, logName.length()-5);
        logName += ".log";
    }
```

Testutskrifter

Alle vil gjøre feil under arbeidet med kompilatoren. For enklere å oppdage feilene når de skjer, skal vi bygge inn ulike testutskrifter som brukeren enkelt kan slå på:

Opsjon	Hva dumpes?	Del
-logC	Kodegenereringen	2
-logP	Parseringen	1
-logS	Skanneren	0
-logT	Lagret parseringstre	1

> rusc -logS mini.rusc

```
   1: /* Program som leser et tall v
   2:    og skriver ut v+1. */
   3:
   4: # Hovedprogrammet:
   5: int main ()
Scanner:  intToken
Scanner:  nameToken main
Scanner:  leftParToken
Scanner:  rightParToken
   6: {
Scanner:  leftCurlToken
   7:    int v;
Scanner:  intToken
Scanner:  nameToken v
Scanner:  semicolonToken
   8:
   9:    /* Les data: */
  10:    v = getint() + 1;
Scanner:  nameToken v
Scanner:  assignToken
Scanner:  nameToken getint
Scanner:  leftParToken
Scanner:  rightParToken
Scanner:  addToken
Scanner:  numberToken 1
Scanner:  semicolonToken
  11:    /* Skriv svaret: */
  12:    putint(v);   putchar(10);
Scanner:  nameToken putint
Scanner:  leftParToken
Scanner:  nameToken v
Scanner:  rightParToken
Scanner:  semicolonToken
Scanner:  nameToken putchar
Scanner:  leftParToken
Scanner:  numberToken 10
Scanner:  rightParToken
Scanner:  semicolonToken
  13: }
Scanner:  rightCurlToken
Scanner:  eofToken
Scanner:  eofToken
```

- Siden utskriften på forrige side kommer fra to kilder, vil flettingen av den kunne variere.

NB!

Variasjoner i fletting er helt normalt og må forventes.

- Når hele programmet er lest, vil skanneren bare returnere **eofToken**.

For å sikre loggfilen, må den lukkes etter hver utskrift.

```
private static void writeLogLine(String data) {
    try {
        PrintWriter log = (nLogLines==0 ? new PrintWriter(logName) :
            new PrintWriter(new FileOutputStream(logName,true)));
        log.println(data);  ++nLogLines;
        log.close();
    } catch (FileNotFoundException e) {
        Error.error("Cannot open log file " + logName + "!");
    }
}
```

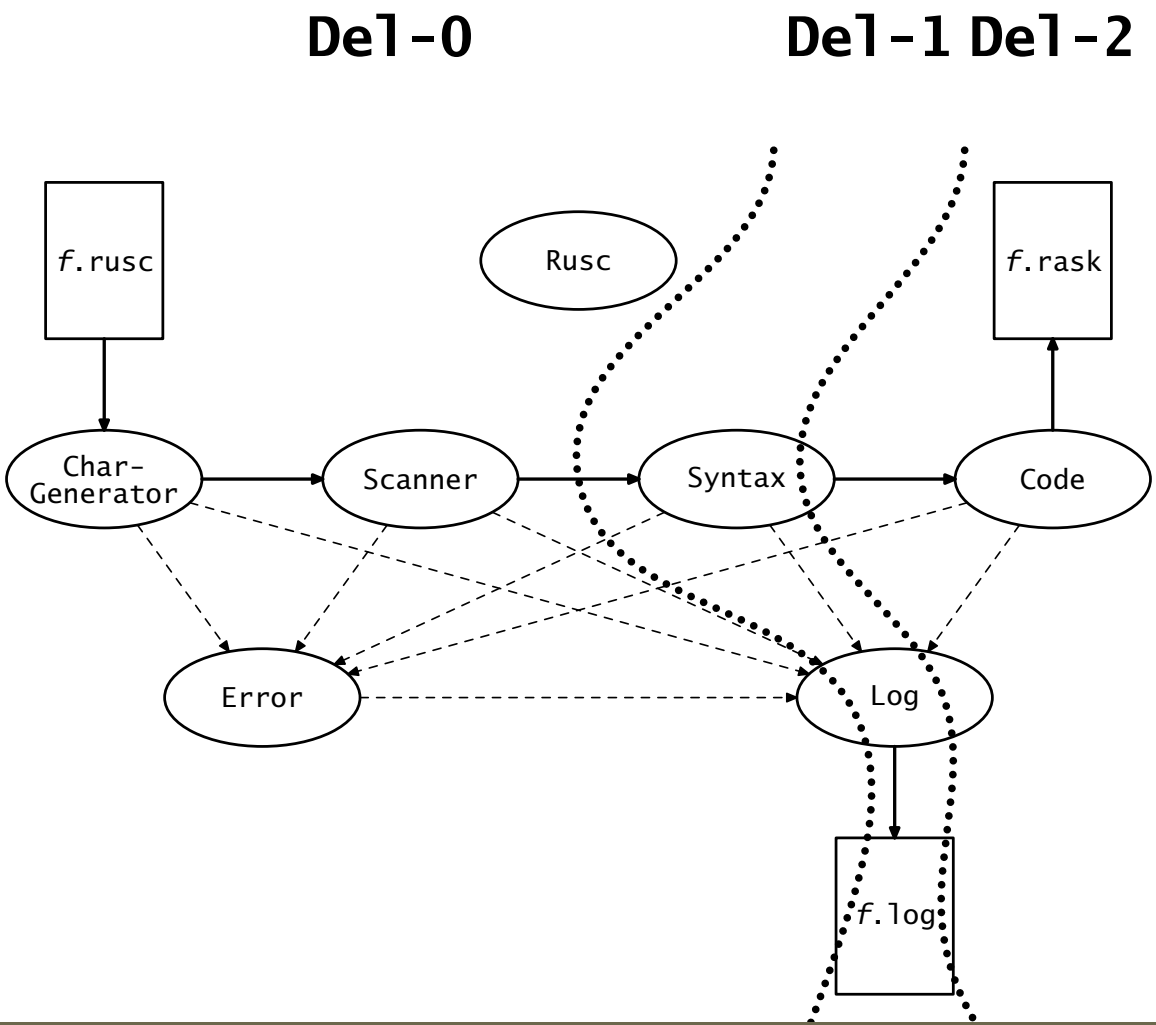
I del 0 skal vi sjekke CharGenerator og Scanner:

```
public static void noteSourceLine(int lineNum, String line) {
    if (! doLogParser && ! doLogScanner) return;

    //-- Må endres i del 0:
}

public static void noteToken(Token t) {
    if (! doLogScanner) return;

    //-- Må endres i del 0:
}
```

Feilmeldinger må med i loggen (om det er noen):

```
public static void noteError(String message) {
    if (nLogLines > 0)
        writeLogLine(message);
}
```

Prosjektet


Del-0
Del-1 Del-2
f.rusc
Rusc
f.rask
Char-
Generator
Scanner
Syntax
Code
Error
Log
f.log

Hvordan komme i gang

Slik starter man å jobbe med koden:

Hent inf2100-oblig.zip fra kursets nettside.

Gjør så:

```
unzip inf2100-oblig.zip
cd inf2100
make
java -jar Rusc.jar minfil.rusc
```

Siste innspill

- Del-0 skal fungere med opsjonen –testscanner.
- Les kompendiet!
- Les Java-koden!
- Det er ikke lov å endre basiskoden.
- Alt skal programmeres fra bunnen av. Det er altså ikke lov å bruke annet fra Java-biblioteket enn java.io (og *Tokenizer er heller ikke lov).
- Gruppelærerne er der for å hjelpe dere.
- Begynn i tide!