Chapter 1

Oblig 1

Course „Compiler Construction“
Martin Steffen
Spring 2019

Section

Compila 19

Chapter 1 „Oblig 1“
Course „Compiler Construction“
Martin Steffen
Spring 2019

Oblig 1

- material (also for oblig 2) based on previous years, including contributions from Eyvind W. Axelsen, Henning Berg, Fredrik Sørensen, and others
- see also the course web-page, containing links to “resources”

Goal (of oblig 1)

Parsing

Determine if programs written in *Compila 19* are syntactically correct:

- scanner
- parser

Rest

- first part of a compiler, oblig 2 will add to it
- language spec provided separatly

Learning outcomes

- using tools for parser/scanner generation
  - JFlex
  - CUP
- variants of a grammar for the same languages
  - transforming one form (EBNF) to another (compatible with the used tools)
  - controlling precedence and associativity
- designing and implementing an AST data structure
  - using the parsing tools to build such trees
  - pretty-printing such trees

Compila language at a glance

```
program MyProgram
begin
   struct complex {      // record data type, but
     re: float;          // no subtyping, polymorphism...
     im: float
   }
end;

proc add (a: complex, b: complex) : complex
begin
   var retval : complex;
in
   retval :=  new complex;
   retval.re := a.re + b.re;
   retval.im := a.im + b.im;
   return retval
end;
proc main()              // execution start here
begin
  var c1: complex;
  var c2: complex:
  result := add (c1,c2);
  ...
  return
end
end
```

Another glance

```
proc swap (a: ref(int), b: ref(int))  // passed as reference
begin
   var tmp: int;
   tmp := deref(a);                   // dereferencing
   deref(a) := deref(b);              // deref can be used both
   deref(b) := tmp                    // left and right of
                                      // an assignment.
end;
```

Grammar (1): declarations

```
PROGRAM          -> "program" NAME "begin" [ DECL {";" DECL}]
"end"

DECL             -> VAR_DECL | PROC_DECL | REC_DECL

VAR_DECL         -> "var" NAME ":" TYPE

PROC_DECL        -> "proc" NAME
                    "(" [ PARAMFIELD_DECL { "," PARAMFIELD_DECL } ]
                    [ ":" TYPE ]
                    "begin" [DECL{";" DECL}] "in" STMT_LIST "end"


STMT_LIST        -> [STMT {";" STMT}]
```

Grammar (2): declarations

```
REC_DECL              -> "struct" NAME "{" [ PARAMFIELD_DECL
                                             {";" PARAMFIELD_DECL }]
"}"

PARAMFIELD_DECL        -> NAME ":" TYPE

EXP                   -> EXP LOG_OP EXP
                       | "not" EXP
                       | EXP REL_OP EXP
                       | EXP ARIT_OP EXP
                       | LITERAL
                       | CALL_STMT
                       | "new" NAME
                       | VAR
                       | REF_VAR
                       | DEREF_VAR
                       | "(" EXP ")"


REF_VAR               -> "ref" "(" VAR ")"
DEREF_VAR             -> "deref" "(" VAR ")" | "deref" "(" DEREF_VAR ")"

VAR                   -> NAME | EXP "." NAME

LOG_OP                -> "&&" | "||"
```

Grammar (3): statements and types

```
ARIT_OP         -> "+" | "-" | "*" | "/" | "^"

LITERAL         -> FLOAT_LITERAL | INT_LITERAL | STRING_LITERAL
                 | "true" | "false" | "null"

STMT            -> ASSIGN_STMT
                 | IF_STMT
                 | WHILE_STMT
                 | RETURN_STMT
                 | CALL_STMT

ASSIGN_STMT     -> VAR ":=" EXP | DEREF_VAR ":=" EXP

IF_STMT         -> "if" EXP "then" { STMT_LIST }
                   [ "else"  { STMT_LIST } ] "fi"
WHILE_STMT      -> "while" EXP "do"  { STMT_LIST  } "od"

RETURN_STMT     -> "return" [ EXP ]

CALL_STMT       -> NAME "(" [ EXP { "," EXP } ] ")"

TYPE            -> "float" | "int" | "string" | "bool" | NAME
                 | "ref" "(" TYPE ")"
```

Section

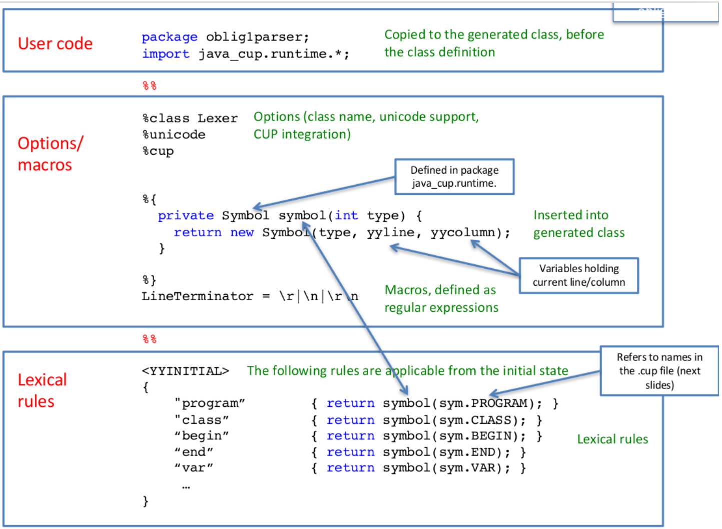
Tools

Chapter 1 „Oblig 1“
Course „Compiler Construction“
Martin Steffen
Spring 2019

Tools: JFlex

- scanner generator (or lexer generator) tool
  - input: lexical specification
  - output: scanner program in Java
- lexical spec written as `.lex` file
- consists of 3 parts
  - user code
  - options and macros
  - lexical rules

Sample lex code



User code

```
package oblig1parser;
import java_cup.runtime.*;
```

Copied to the generated class, before the class definition

```
%%
```

Options/ macros

```
%class Lexer
%unicode
%cup
```

Options (class name, unicode support, CUP integration)

```
%{
  private Symbol symbol(int type) {
    return new Symbol(type, yyline, yycolumn);
  }

%}
LineTerminator = \r|\n|\r\n
```

Defined in package java_cup.runtime.

Inserted into generated class

Variables holding current line/column

Macros, defined as regular expressions

```
%%
```

Lexical rules

```
<YYINITIAL>
{
    "program"        { return symbol(sym.PROGRAM); }
    "class"          { return symbol(sym.CLASS); }
    "begin"          { return symbol(sym.BEGIN); }
    "end"            { return symbol(sym.END); }
    "var"            { return symbol(sym.VAR); }
     …
}
```

The following rules are applicable from the initial state

Refers to names in the .cup file (next slides)

Lexical rules

CUP: Construction of useful parsers (for Java)

- a tool to easily (ymmv) generate *parsers*
- reads tokes from the scanner using `next_token()`
- the `%cup` option (previous slide) makes that work

Input

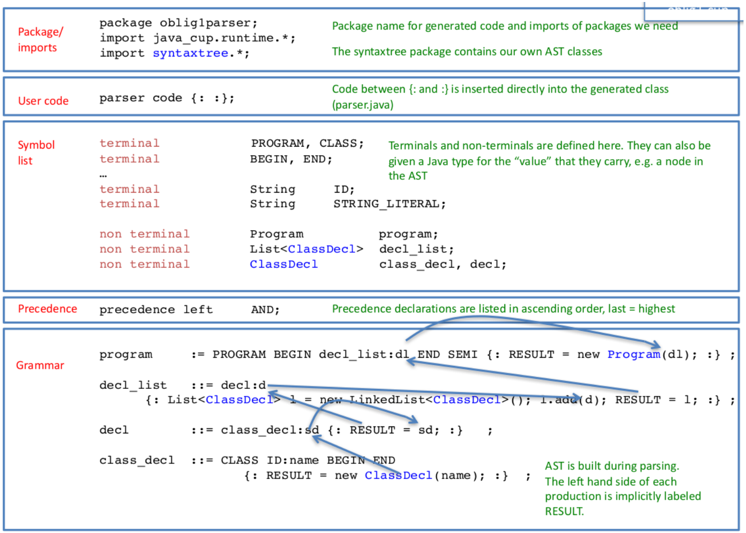
grammar in BNF with action code

```
var_decl ::= VAR ID:name COLON type:vtype
 {: RESULT =  new VarDecl(name, vtype); :};
```

Rest

- output: parser program (in Java)

Sample CUP code



Package/ imports

```
package oblig1parser;
import java_cup.runtime.*;
import syntaxtree.*;
```

Package name for generated code and imports of packages we need

The syntaxtree package contains our own AST classes

User code

```
parser code {: :};
```

Code between {: and :} is inserted directly into the generated class (parser.java)

Symbol list

```
terminal                  PROGRAM, CLASS;
terminal                  BEGIN, END;
…
terminal                  String     ID;
terminal                  String     STRING_LITERAL;

non terminal              Program          program;
non terminal              List<ClassDecl>  decl_list;
non terminal              ClassDecl        class_decl, decl;
```

Terminals and non-terminals are defined here. They can also be given a Java type for the "value" that they carry, e.g. a node in the AST

Precedence

```
precedence left      AND;
```

Precedence declarations are listed in ascending order, last = highest

Grammar

```
program     := PROGRAM BEGIN decl_list:dl END SEMI {: RESULT = new Program(dl); :} ;

decl_list   ::= decl:d
       {: List<ClassDecl> l = new LinkedList<ClassDecl>(); l.add(d); RESULT = l; :} ;

decl        ::= class_decl:sd {: RESULT = sd; :}   ;

class_decl  ::= CLASS ID:name BEGIN END
                  {: RESULT = new ClassDecl(name); :}  ;
```

AST is built during parsing. The left hand side of each production is implicitly labeled RESULT.

Build tool: ant

- Java-based build tool (think "make")
- config in `build.xml`
- can contain different targets

typical general targets

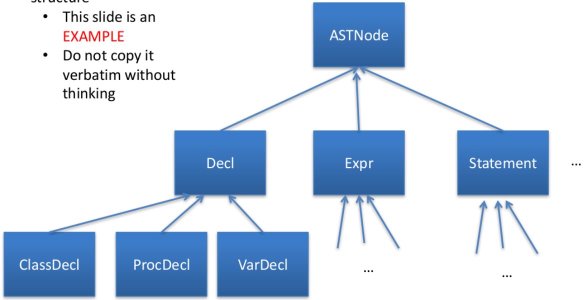
- test
- clean
- build
- run

Rest

- supplied configuration should take care of calling `jflex`, `cup`, and `javadoc` for you

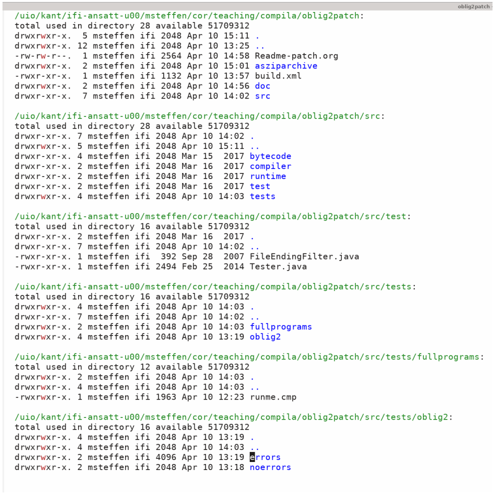
AST data structure

- Make a reasonable structure
  - This slide is an EXAMPLE
  - Do not copy it verbatim without thinking



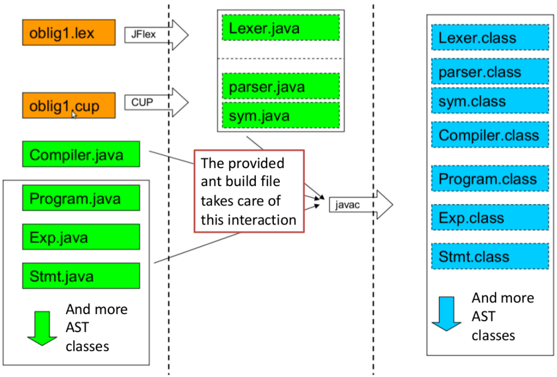
Overview over the directory + first steps

- see the Readme at/from the `github.uio.no`



```
/uio/kant/ifi-ansatt-u00/msteffen/cor/teaching/compila/oblig2patch:
total used in directory 28 available 51709312
drwxrwxr-x.  5 msteffen ifi 2048 Apr 10 15:11 .
drwxrwxr-x. 12 msteffen ifi 2048 Apr 10 13:25 ..
-rw-rw-r--.  1 msteffen ifi 2564 Apr 10 14:58 Readme-patch.org
drwxrwxr-x.  2 msteffen ifi 2048 Apr 10 15:01 asziparchive
-rwxr-xr-x.  1 msteffen ifi 1132 Apr 10 13:57 build.xml
drwxrwxr-x.  2 msteffen ifi 2048 Apr 10 14:56 doc
drwxr-xr-x.  7 msteffen ifi 2048 Apr 10 14:02 src

/uio/kant/ifi-ansatt-u00/msteffen/cor/teaching/compila/oblig2patch/src:
total used in directory 28 available 51709312
drwxr-xr-x. 7 msteffen ifi 2048 Apr 10 14:02 .
drwxrwxr-x. 5 msteffen ifi 2048 Apr 10 15:11 ..
drwxr-xr-x. 4 msteffen ifi 2048 Mar 15  2017 bytecode
drwxr-xr-x. 2 msteffen ifi 2048 Mar 16  2017 compiler
drwxr-xr-x. 2 msteffen ifi 2048 Mar 16  2017 runtime
drwxr-xr-x. 2 msteffen ifi 2048 Mar 16  2017 test
drwxrwxr-x. 4 msteffen ifi 2048 Apr 10 14:03 tests

/uio/kant/ifi-ansatt-u00/msteffen/cor/teaching/compila/oblig2patch/src/test:
total used in directory 16 available 51709312
drwxr-xr-x. 2 msteffen ifi 2048 Mar 16  2017 .
drwxr-xr-x. 7 msteffen ifi 2048 Apr 10 14:02 ..
-rwxr-xr-x. 1 msteffen ifi  392 Sep 28  2007 FileEndingFilter.java
-rwxr-xr-x. 1 msteffen ifi 2494 Feb 25  2014 Tester.java

/uio/kant/ifi-ansatt-u00/msteffen/cor/teaching/compila/oblig2patch/src/tests:
total used in directory 16 available 51709312
drwxrwxr-x. 4 msteffen ifi 2048 Apr 10 14:03 .
drwxr-xr-x. 7 msteffen ifi 2048 Apr 10 14:02 ..
drwxrwxr-x. 2 msteffen ifi 2048 Apr 10 14:03 fullprograms
drwxrwxr-x. 4 msteffen ifi 2048 Apr 10 13:19 oblig2

/uio/kant/ifi-ansatt-u00/msteffen/cor/teaching/compila/oblig2patch/src/tests/fullprograms:
total used in directory 12 available 51709312
drwxrwxr-x. 2 msteffen ifi 2048 Apr 10 14:03 .
drwxrwxr-x. 4 msteffen ifi 2048 Apr 10 14:03 ..
-rwxrwxr-x. 1 msteffen ifi 1963 Apr 10 12:23 runme.cmp

/uio/kant/ifi-ansatt-u00/msteffen/cor/teaching/compila/oblig2patch/src/tests/oblig2:
total used in directory 16 available 51709312
drwxrwxr-x. 4 msteffen ifi 2048 Apr 10 13:19 .
drwxrwxr-x. 4 msteffen ifi 2048 Apr 10 14:03 ..
drwxrwxr-x. 2 msteffen ifi 4096 Apr 10 13:19 errors
drwxrwxr-x. 2 msteffen ifi 2048 Apr 10 13:18 noerrors
```

Building: putting it together



Section

Official

Chapter 1 „Oblig 1“
Course „Compiler Construction“
Martin Steffen
Spring 2019

Deadline

Deadline

Friday 15. 03. 2019, 23:59

Rest

- don't miss the deadline
- for extensions, administration needs to agree (`studadm`), contact them if sick etc
- even if not 100% finished
  - deliver what you have
  - contact early when problems arise

Deliverables

- see also the “handout”

Deliverables (1)

- working parser
  - parse the supplied sample programs
  - printout the resulting AST
- two grammars (two `.cup`-files)
  - one unambiguious
  - one ambiguous, where ambibuities resolved through precedence declations in *CUP*, e.g.

```
precendence left AND;
```

Deliverables

Deliverables (2)

- report (with name(s) and UiO user name(s)
- discussion of the solution (see handout for questions)
- in particular: comparison of the two grammars
- “Readme”

Rest

- the code must *build* (with ant) and run
- test it on the UiO RHEL (linux) platform

Ask

If problems, ask in time (NOT Friday at the deadline)

Hand-in procedure

- this year we try *git*
- `https://github.uio.no` resp. `https://github.uio.no/msteffen/compila`
- you need
  - a login
  - send me emails that you want to do oblig (+ potential partner) ⇒ I tell you group number
  - create a project `compila<n>` (n = group number)
  - add collaborator + (at some point me)
- see also the handout

Chapter 2

Oblig2

Course „Compiler Construction“
Martin Steffen
Spring 2019

Goal

1. semantic analysis, as far as
   - typing is concerned (“static semantics”)
   - other coditions (no duplicate declaration etc)
2. code generation for `compila19` (ish) programs

Last time (O1)

Syntactic analysis

- lexer (scanner)
- parser
- abstract syntax tree

this time: continue with your previous deliv. (and repos)

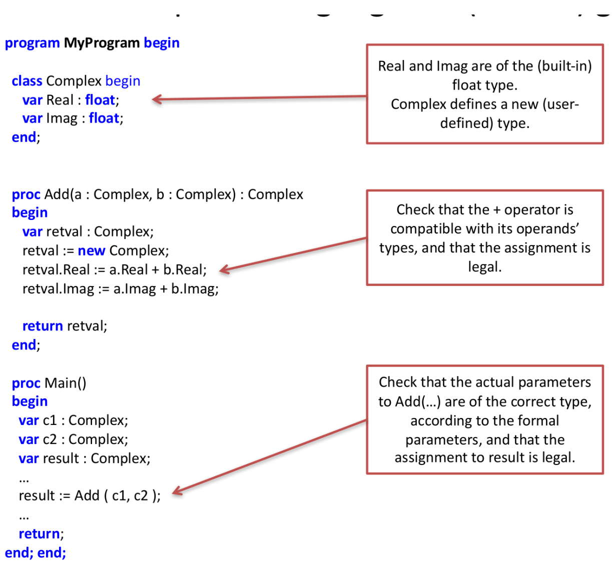
Learning outcome

- understand type checking, implementing a simple variant
- understand (simple form of) bytecode and how to generate it from "source code" (as AST)
- extend an existing compiler code base with new functionality

Semantic analysis & type checking

- parser / context-free grammars
  - not powerful enough
  - cannot check all (static) properties of a language spec
- => extend the front-end by a type checker
  - use the AST classes of last time
  - add type checking code
  - allowed to make changes or adaptations if advantagous.

Another glance at compila19



```
program MyProgram begin

 class Complex begin
   var Real : float;
   var Imag : float;
 end;


 proc Add(a : Complex, b : Complex) : Complex
 begin
   var retval : Complex;
   retval := new Complex;
   retval.Real := a.Real + b.Real;
   retval.Imag := a.Imag + b.Imag;

   return retval;
 end;

 proc Main()
 begin
  var c1 : Complex;
  var c2 : Complex;
  var result : Complex;
  ...
  result := Add ( c1, c2 );
  ...
  return;
end; end;
```

Real and Imag are of the (built-in) float type. Complex defines a new (user-defined) type.

Check that the + operator is compatible with its operands' types, and that the assignment is legal.

Check that the actual parameters to Add(...) are of the correct type, according to the formal parameters, and that the assignment to result is legal.

NB: 2019: structs, not classes

Type checking for conditionals

- as "inspiration", details may vary

```
  class IfStatement extends Statement {
  ...
    public void typeCheck(){
      String condType = condition.get.Type ();
      if (condType != "bool") {
         throw new TypeException("condition in an if
           statement must be of type bool")
      }
}
```

Type checking: assignments

```
class Assignment extends Statement {
...
  public void typeCheck() {
    String varType = var.getType();
    String expType = exp.getType();
    if (varType != expType &&
        !isAssigmentCompatible(varType,expType){
                   throw new TypeException("Cannot assig
                   " from " + expType);
  }
}
```

Code generation

- byte code API and operations are described in the document “Interpreter and bytecode for INF5110”
- Task: add bytecode generation methods to your AST classes for instance

```
Ast.Node.GenerateCode(...)
```

- again: if adaptations of the AST are called for or useful, go for it...

Code generation: limitations

- interpreter and byte code library somewhat limited
  - cannot express full compila 19
  - no block structure
  - no reference types
- your delivery should support generating correct bytecode for the `compila` 19 source code file `runme.cmp`

Code generation: creating a procedure

```
CodeFile codeFile = new CodeFile();
// add the procedure by name first
codeFile.addProcedure("Main")
// then define it
CodeProcedure main = new
   CodeProcedure("Main", VoidType,TYPE, codeFile);
main.addInstruction( new RETURN());
//then update it in the code file
codeFile.updateProcedure(main);
```

Code generation: assignment

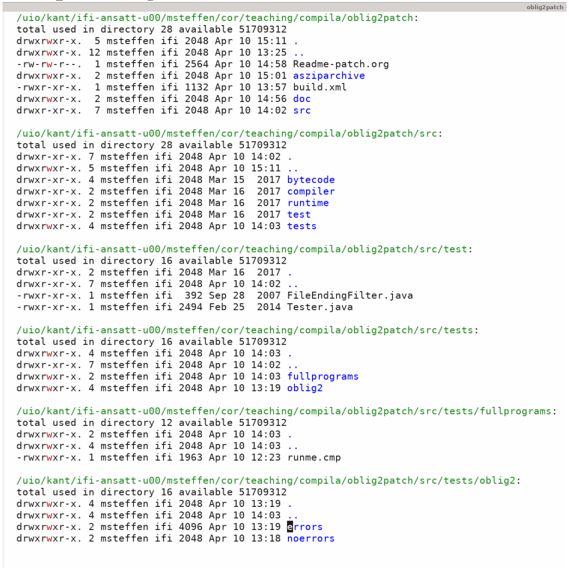
```
//1: proc add(a: int, b : int ) : int {
//2: var res : int;
//3: res := a + b; // only bytecode for this line
//4: return res;
//5: }

// push a onto the stack
proc.addInstruction(new LOADLOCAL(proc.variableNumber("a")));
// push b onto the stack
proc.addInstruction(new LOADLOCAL(proc.variableNumber("b")));
// perform addition with arguments on the stack
proc.addInstruction(new ADD());
// pop result from stack, and store it in variable res
proc.addInstruction(new
        STORELOCAL(proc.variableNumber("res")));
```

Testing

- bunch of test files, for testing the *type checker*
- preferable: make `ant test` workable
- test files ending with `fail` containt a syntactically correct but erronous program (erroneous as the type system or generally the semantic phase is concerned)
- => compiler returns error code 2 for semantic failure

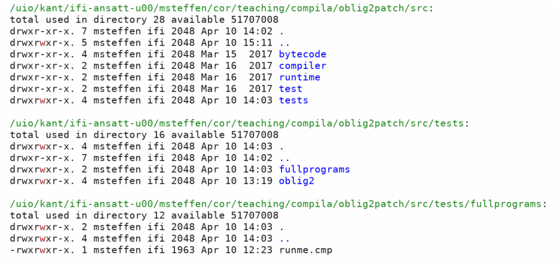
Provided source code

https://github.uio.no/msteffen/compila



```
/uio/kant/ifi-ansatt-u00/msteffen/cor/teaching/compila/oblig2patch:
total used in directory 28 available 51709312
drwxrwxr-x.  5 msteffen ifi 2048 Apr 10 15:11 .
drwxrwxr-x. 12 msteffen ifi 2048 Apr 10 13:25 ..
-rw-rw-r--.  1 msteffen ifi 2564 Apr 10 14:58 Readme-patch.org
drwxrwxr-x.  2 msteffen ifi 2048 Apr 10 15:01 asziparchive
-rwxr-xr-x.  1 msteffen ifi 1132 Apr 10 13:57 build.xml
drwxrwxr-x.  2 msteffen ifi 2048 Apr 10 14:56 doc
drwxr-xr-x.  7 msteffen ifi 2048 Apr 10 14:02 src

/uio/kant/ifi-ansatt-u00/msteffen/cor/teaching/compila/oblig2patch/src:
total used in directory 28 available 51709312
drwxr-xr-x. 7 msteffen ifi 2048 Apr 10 14:02 .
drwxrwxr-x. 5 msteffen ifi 2048 Apr 10 15:11 ..
drwxr-xr-x. 4 msteffen ifi 2048 Mar 15  2017 bytecode
drwxr-xr-x. 2 msteffen ifi 2048 Mar 16  2017 compiler
drwxr-xr-x. 2 msteffen ifi 2048 Mar 16  2017 runtime
drwxr-xr-x. 2 msteffen ifi 2048 Mar 16  2017 test
drwxrwxr-x. 4 msteffen ifi 2048 Apr 10 14:03 tests

/uio/kant/ifi-ansatt-u00/msteffen/cor/teaching/compila/oblig2patch/src/test:
total used in directory 16 available 51709312
drwxr-xr-x. 2 msteffen ifi 2048 Mar 16  2017 .
drwxr-xr-x. 7 msteffen ifi 2048 Apr 10 14:02 ..
-rwxr-xr-x. 1 msteffen ifi  392 Sep 28  2007 FileEndingFilter.java
-rwxr-xr-x. 1 msteffen ifi 2494 Feb 25  2014 Tester.java

/uio/kant/ifi-ansatt-u00/msteffen/cor/teaching/compila/oblig2patch/src/tests:
total used in directory 16 available 51709312
drwxrwxr-x. 4 msteffen ifi 2048 Apr 10 14:03 .
drwxr-xr-x. 7 msteffen ifi 2048 Apr 10 14:02 ..
drwxrwxr-x. 2 msteffen ifi 2048 Apr 10 14:03 fullprograms
drwxrwxr-x. 4 msteffen ifi 2048 Apr 10 13:19 oblig2

/uio/kant/ifi-ansatt-u00/msteffen/cor/teaching/compila/oblig2patch/src/tests/fullprograms:
total used in directory 12 available 51709312
drwxrwxr-x. 2 msteffen ifi 2048 Apr 10 14:03 .
drwxrwxr-x. 4 msteffen ifi 2048 Apr 10 14:03 ..
-rwxrwxr-x. 1 msteffen ifi 1963 Apr 10 12:23 runme.cmp

/uio/kant/ifi-ansatt-u00/msteffen/cor/teaching/compila/oblig2patch/src/tests/oblig2:
total used in directory 16 available 51709312
drwxrwxr-x. 4 msteffen ifi 2048 Apr 10 13:19 .
drwxrwxr-x. 4 msteffen ifi 2048 Apr 10 14:03 ..
drwxrwxr-x. 2 msteffen ifi 4096 Apr 10 13:19 errors
drwxrwxr-x. 2 msteffen ifi 2048 Apr 10 13:18 noerrors
```

Provided source code



```
/uio/kant/ifi-ansatt-u00/msteffen/cor/teaching/compila/oblig2patch/src:
total used in directory 28 available 51707008
drwxr-xr-x. 7 msteffen ifi 2048 Apr 10 14:02 .
drwxrwxr-x. 5 msteffen ifi 2048 Apr 10 15:11 ..
drwxr-xr-x. 4 msteffen ifi 2048 Mar 15  2017 bytecode
drwxr-xr-x. 2 msteffen ifi 2048 Mar 16  2017 compiler
drwxr-xr-x. 2 msteffen ifi 2048 Mar 16  2017 runtime
drwxr-xr-x. 2 msteffen ifi 2048 Mar 16  2017 test
drwxrwxr-x. 4 msteffen ifi 2048 Apr 10 14:03 tests

/uio/kant/ifi-ansatt-u00/msteffen/cor/teaching/compila/oblig2patch/src/tests:
total used in directory 16 available 51707008
drwxrwxr-x. 4 msteffen ifi 2048 Apr 10 14:03 .
drwxr-xr-x. 7 msteffen ifi 2048 Apr 10 14:02 ..
drwxrwxr-x. 2 msteffen ifi 2048 Apr 10 14:03 fullprograms
drwxrwxr-x. 4 msteffen ifi 2048 Apr 10 13:19 oblig2

/uio/kant/ifi-ansatt-u00/msteffen/cor/teaching/compila/oblig2patch/src/tests/fullprograms:
total used in directory 12 available 51707008
drwxrwxr-x. 2 msteffen ifi 2048 Apr 10 14:03 .
drwxrwxr-x. 4 msteffen ifi 2048 Apr 10 14:03 ..
-rwxrwxr-x. 1 msteffen ifi 1963 Apr 10 12:23 runme.cmp
```

- Java
  - `compiler`: updated compiler class
  - `test`: some code for performing tests
  - `bytecode`: classes for constructing bytecode
  - `runtime`: rte for executing the byte code
- Compila
  - `tests`: some test files (including `runme.cmp`)

Deadline

Deadline

12th May 2019

Note: end of semester, and I need to report the ones passing the oblig some time before the exam.

delivs

- working type checker
- code generator (test with `runme.cmp`)
- report (including your name(s) etc.
  - discussion of your solution, choices you made, assumptions you rely on
  - printout of a test run (can be also checked in into the repos, but it n needs to be mentioned where it is)
  - printout of the bytecode from `runme.cmp` (with a target like `ant list-runme`)
  - solution must "build" and be "testable" (typically via `ant`)

Section

References

Chapter 2 „Oblig2“
Course „Compiler Construction“
Martin Steffen
Spring 2019

References I