Universitetet i Oslo
Institutt for Informatikk

Reliable systems
Martin Steffen

# INF 5110: Compiler construction

Spring 2021 — **Oblig 2** — 7. 04. 2021

**Issued: 7. 04. 2021**

## 1 Official

The **deadline/frist** for the second oblig is

**14. May 2021**

## 2 What and how to hand in

### 2.1 Git

You will continue with your group's git-repos you used in the first oblig (unless really convincing reasons speak against it). Basically, you continue with your previous code, add the new functionality, push a solution before the deadline, and inform me when it's done so that I can update. It's important to tell me, as I don't want to repeatedly update in the hope that it's done.

If a change in arrangement is needed (merge of groups, or a split of groups), you need to ask for that re-arrangement (not just that on the day of the deadline it's "announced" that there is now a new group ...).

See also the *Readme* of the *"patch"* under

`https://github.uio.no/msteffen/compila/tree/master/oblig2patch`

### 2.2 What to include into a solution

As before, it should be an appropriately commented repos, solving the tasks of oblig 2. In particular needed is (basically as before)

- A top-level *Readme-file*[1] containing
  - containing names and emails of the authors
  - instructions how to build the compiler and how to run it.
  - test-output for running the compiler on `compila.cmp` as input

[1] Many did a `Readme.md` which is a good format.

- of course, all code needed to run your solution
- the Java-classes for the syntax-tree
- the build-script `build.xml` (adapted)

Of course, the old code (for lex and yacc-based parsing) is still needed. It's not needed that both versions of the grammar, required for oblig 1, are still supported, one working version is enough.

# 3 Purpose and goal

The goal of the task is to collect more practical experience implementing a compiler, in particular, a taste of phases after parsing. It's only a taste, as we don't have the time to get a full-scale compiler on its feet. The language we are compiling is (as before) described in the *compila 20 language specification.* This time, also the later sections about type checking etc, what were irrelevant for oblig 1, specify the scope of the task as far as the language features are concerned.

Testing becomes more important than in oblig 1. It's *necessary* that a solution is equipped

with "automatic test-cases"

That can be done (as before) via ant targets. Those tests have to be executable on the RHEL linux pool at the university.[2]

# 4 Tools

The tools are basically the same as for the previous oblig, and typically you will continue anyway with the previous set-up.

# 5 Task more specifically: Type checking and code generation

The task is to extend the parser and AST generation with type checking and code generation. The rules governing the type checking and other restrictions are described in the language specification already (in the later sections). The "semantics" is *not* specified, but the language is so simple that it should basically be clear what a compila program is supposed to do.

The target "platform" is described in a separate document (which was already made available as part of the git-repos). It's also browsable under

`https://github.uio.no/msteffen/compila/tree/master/doc/bytecodeinterpreter`

### Tests

The tests that need to be successfully run for oblig 2 are

1. testing the type checker resp. semantic analysis

[2]That should actually not be a big restriction, as Java (and the task) is to a big extent platform independent ("write once, run everywhere" ...). Nonetheless: Based on experience with the earlier years (this year actually no problems occured): it's advised to make this "test" setup early on (not after the deadline), to design the code *with the goal that it runs also at a different place than one's own platform* and to test that this goal is actually met. The reason for that "testability" requirement is that correction will again not be based on reading much code from my side, but in first approximation: running the test. In that sense, it's also not of primary importance, whether it's `ant` or perhaps *make* or some script. Important is, that I can execute it by invoking a simple command. I don't have the time to figure out how one particular solution is configured, started, etc. I don't even want to look around and try whether I find a `main` method somewhere...

2. testing the code coge generator

The tests are located as follows relative to the `oblig2patch`-directory

`./src/tests/semanticanalysis/`

`./src/tests/fullprograms/runme.cmp`

You may place them inside the *you* src-directory (and add then to *your* repository).

## Patch

Obtain the patch (as zip-archive) under

`https://github.uio.no/msteffen/compila/tree/master/oblig2patch/oblig2patch.zip`

or via an updated clone of the course repos. Read also the Readme.org there, there is more info about how to start. Note: when you cloned or downloaded the repository in perhaps March, there had been already some *oblig2patch* as part of the repos. Also that should be more or less usable, it reflects the 2020 version. Not much has changed since then (mostly the documentation like this file handout here and the slides, the actualy "content" is basically the same