Chapter 1

Oblig 2

Course “Compiler Construction”

Martin Steffen

Spring 2020

Goal

1. semantic analysis, as far as
   - typing is concerned ("static semantics")
   - other coditions (no duplicate declaration etc)
2. code generation for `compila20` (ish) programs

Last time (O1)

Syntactic analysis

- lexer (scanner)
- parser
- abstract syntax tree

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

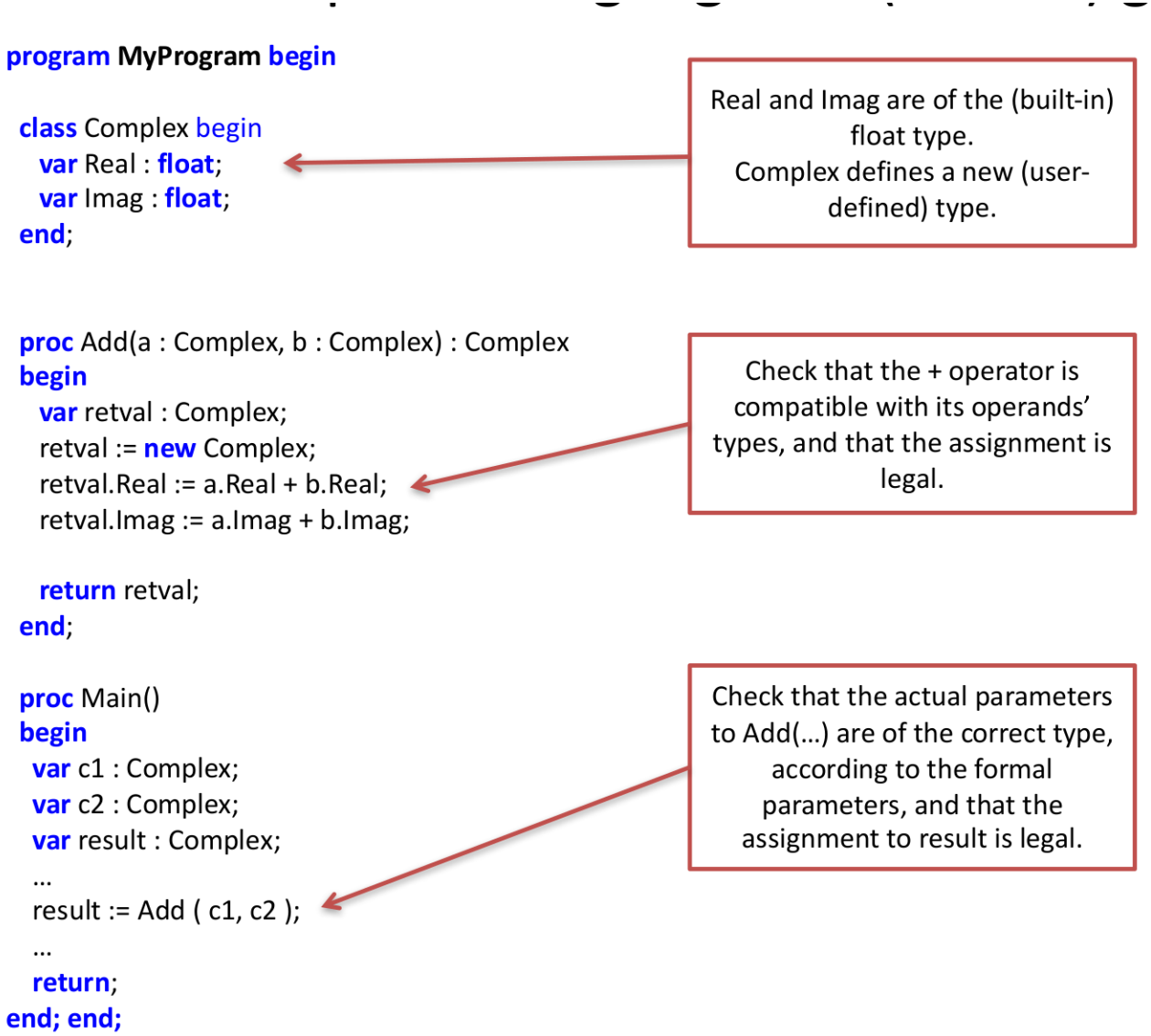
Learning outcome

- 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 compila20



NB: 2020: 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 assi
                  " 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...
- some people did visitors for ast-printing, one can also (re-)use the visitor pattern

Code generation: limitations

- interpreter and byte code library somewhat limited
  - cannot express full compila 20
  - no block structure
  - no reference types
- your delivery should support generating correct bytecode for the `compila` 20 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")));
```

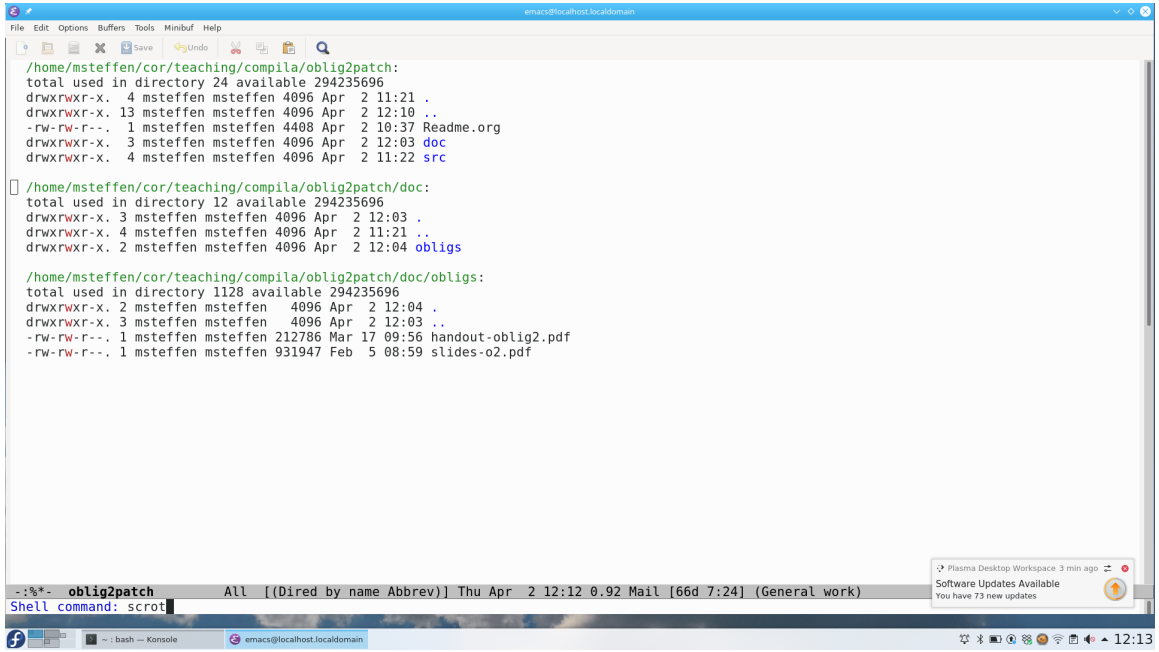
Testing

- bunch of test files, for testing the *type checker*
- preferable: make `ant test` workable
- test files inside `./tests/semanticanalysis/errors/` (and with `fail` in the filename) contain 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 (patch)

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

Provided documentation (patch)



- 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 2020

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 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`)