Obligatorisk oppgave 2 – Litt flere detaljer om semantikksjekk og kodegenerering

Fredrik Sørensen
OMS-gruppen, IfI

Viktig

- Ny patch (patch_oblig2.zip) er lagt ut (15/4)
- Oblig 1 vil bli rettet **denne** uken
  - Sjekk om det er registrert at den er levert (https://wwws.ifi.uio.no/)
- Les listen med endringer og beskjeder!
- Det vil helt sikkert bli flere revisjoner av koden
- Jeg deler ut notat som beskriver byte-kode og interpreter i dag. Endelig versjon kommer på kurssiden når vi har silt ut noen flere feil.

Rettet i koden

- Loader.java
  - Feilen var at PUSHBOOL fikk verdien TRUE hver gang fordi jeg tok med en ekstra byte hver gang (short istedenfor byte).
- PUSHBOOL.java
  - Jeg mappet TRUE -> FALSE og FALSE -> TRUE.
- STOREGLOBAL.java
  - Jeg har rette opp litt på utskriften (listing) av byte-koden.
- GETFIELD.java og PUTFIELD.java
  - structNum ble alltid 0 og fieldNum ble overskrevet med verdien til struktNum siden det stod 1 og ikke 3 som indeks.

Semantikksjekk

- Det er nok en fordel å bruke en symboltabell (s. 295 i boken, illustrasjon s. 304)
- Returnere så fort man finner en feil.
  - Kaste Exception
  - Returverdi (true/false, egen klasse, int?)
- Kan være en fordel å lage en abstrakt klasse Type og sjekke typelikhet og lovlig automatisk casting gjennom den.

Forskjellene i semantikken er…

- Ikke blokkstruktur.
  - Den virtuelle maskinen har kun et globalt og lokalt nivå.
- Det er ikke referanseparamtere
  - basisparamterene overføre by-value
  - struktvariablene er pekerverdier og overføres for så vidt også by-value
  - Altså: Det er på sammen måte som i Java.
- Legg merke til at det har blitt brukt litt andre navn, slik som at funksjonene kalles prosedyrer (Procedure).

Å bruke bytecode-biblioteket

```
CodeFile codeFile = new CodeFile();

// Her bygges bytekoden opp ...


byte[] bytecode = codeFile.getBytecode();

DataOutputStream stream =
  new DataOutputStream(
    new FileOutputStream("eks.bin"));

stream.write(bytecode); stream.close();
```

Et eksempel

```
// Make code:
CodeFile codeFile = new CodeFile();

codeFile.addProcedure("print_float");
codeFile.addProcedure("Main");
codeFile.addVariable("myGlobalVar");
codeFile.addProcedure("test");
codeFile.addStruct("Complex");

CodeProcedure main = new CodeProcedure("Main",
  VoidType.TYPE, codeFile);
main.addInstruction(new RETURN());
codeFile.updateProcedure(main);
codeFile.updateVariable("myGlobalVar", new
  RefType(codeFile.structNumber("Complex")));
```

Et eksempel

```
CodeProcedure test = new CodeProcedure("test",
  VoidType.TYPE, codeFile);

test.addParameter("firstPar", FloatType.TYPE);
test.addParameter("secondPar", new
   RefType(test.structNumber("Complex")));
test.addInstruction(new
   LOADLOCAL(test.variableNumber("firstPar")));
test.addInstruction(new
   CALL(test.procedureNumber("print_float")));
test.addInstruction(new RETURN());

codeFile.updateProcedure(test);
```

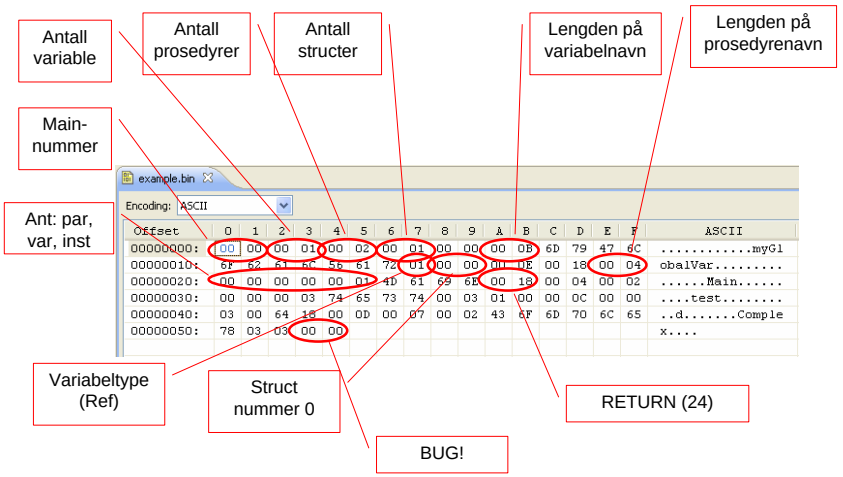
Et eksempel

```
CodeStruct complex = new
  CodeStruct("Complex");

complex.addVariable("Real", FloatType.TYPE);
complex.addVariable("Imag", FloatType.TYPE);
codeFile.updateStruct(complex);

CodeProcedure printFloat = new
   CodeProcedure("print_float", VoidType.TYPE,
   codeFile);
printFloat.addParameter("f", FloatType.TYPE);
codeFile.updateProcedure(printFloat);

codeFile.setMain("Main");
byte[] bytecode = codeFile.getBytecode();
// ... Write the bytes to a file.
```

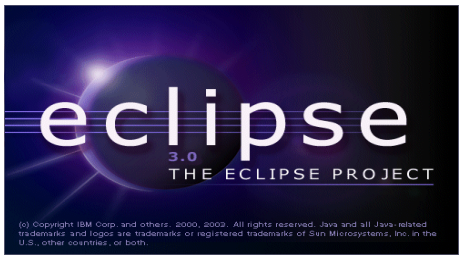
Resultatet

```
Loading from file: ./code-examples/example.bin
Variables:
  0: var Complex myGlobalVar
Procedures:
  0: func void Main()
    0: return
  1: func void test(float 0, Complex 1)
    0: loadlocal 0
    2: call print_float {100}
    4: return
Structs:
  0: Complex
    0: float
    1: float
Constants:
STARTWITH: Main
```

Som bytecode



Og så kikker vi litt i koden



Noen eksempler på tavlen

- Struct-aksess
- Call
- Yttrykk med ’+’ og tilordning
  - var int x;
  - x := 2 + 2;
- Tekst => Klasser i Bytecode-bibliotek => Bytecode-fil => Runtime (Stack)

Kjøre kompilator og VM

- Open Debug Dialog
- Høyreklikk Java Application => New
- Velg klassen Compiler
- Skriv in Arguments
  - Program Arguments, ikke VM Arguments
  - Alle tre filene: *.d, *.ast, *.bin
- Det samme for VirtualMachine
  - Men arguments er bare *.bin (eventuelt “-l” foran)

Husk fristen

Endelig frist **9. mai**

Ta kontakt

- Epost: fredrso@ifi.uio.no
- TLF: 22 85 04 73
- Kontor: 4311-NR (4. etasje)

Obligatorisk veiledning

- Torsdag, 24/4 (hele dagen) og fredag 25/4 (etter 12.00)
- 15 minutter/gruppe
- Gjør avtale med meg
  - ”Første til mølla”-prinsippet
- Dere skal fortelle hva dere har gjort
  - Ikke lag PowerPoint-presentasjon!
  - Vis meg koden
  - Jeg stiller noen spørsmål
- Jeg besvarer alle spørsmål, så vær godt forberedt og dere kan få mye ut av det!

SPØRSMÅL ???