

INF5110

Obligatorisk Oppgave 2

del 2

Andreas Svendsen
SINTEF

23. April 2009

Oversikt

- Tilbakeblikk på oppgaven
- Eksempel på sjekk av semantikk
- Eksempel på kodegenerering
- Nødvendige instruksjoner for IF-noden
- Oppsummering

Informasjon

- Oblig 1 er rettet
 - Sjekk at godkjentlisten er riktig på <https://wwws.ifi.uio.no/>

Oppgaven

- Utvide syntakstre-nodene til også å sjekke semantikk
- Bruke biblioteksklasser til å generere byte-kode
- Teste semantikksjekken mot programmene i testsuiten
- Bare teste byte-kode-genereringen mot eksemplet RunMe.d

Sjekke semantikk

- PROGRAM-noden:
 - Legge alle deklarasjoner til i symboltabellen
 - Sjekke at de ikke er deklartert dobbelt
 - Sjekke semantikken til deklarasjonene
 - Sjekke at det er deklartert en Main-funksjon
 - Kontrollere at den har returverdi VOID
 - Kontrollere at den ikke har parametere

Sjekke semantikk (2)

- FUNCDECL-noden:
 - Legge til en ny blokk i symboltabellen
 - Legge til parametere og deklarasjoner i symboltabellen
 - Sjekke at de ikke er deklartert dobbelt
 - Sjekke semantikken til statements-ene
 - Sjekke at funksjonen enten er deklartert med returverdi eller VOID
 - I dette tilfellet kan typesjekken av returverdien gjøres i RETURN-noden

Sjekke semantikk (3)

- La resten av semantikken sjekkes av de respektive nodene i syntakstreet
 - Tilsvarende PrettyPrint-funksjonen

Generere kode - Start

// Make code:

```
CodeFile codeFile = new CodeFile();  
codeFile.addProcedure("print_float");
```

Alt
starter
her

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

Legge til
globale
deklarasjoner

Generere kode - Main

```
CodeProcedure main = new CodeProcedure("Main",  
    VoidType.TYPE,codeFile);  
main.addInstruction(new RETURN());  
codeFile.updateProcedure(main);
```

Nytt objekt
for en
metode

```
// Oppdatering av global variabel  
codeFile.updateVariable("myGlobalVar", new  
    RefType(codeFile.structNumber("Complex")));
```

Husk å oppdatere
metoder og variable

Generere kode - Procedure

```
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);
```

En variabel
må legges
på stacken
før den kan
brukes som
parameter

Husk RETURN-
instruksjon for metoder

Generere kode - Strukt

```
CodeStruct complex = new CodeStruct("Complex");
complex.addVariable("Real", FloatType.TYPE);
complex.addVariable("Imag", FloatType.TYPE);
codeFile.updateStruct(complex);
```

Generere kode - End

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

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

Husk å sette
hvilken
metode som
er Main

Listing av koden

Loading from file: ./example.bin

Variables:

0: var Complex myGlobalVar

Procedures:

0: func void print_float()

1: func void Main()

0: return

2: func void test(float 0, Complex 1, float 2)

0: loadlocal 0

3: call print_float {0}

6: return

Structs:

0: Complex

0: float

1: float

Constants:

STARTWITH: Main

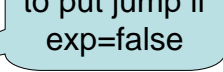
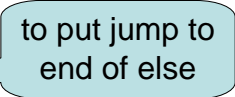
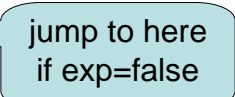
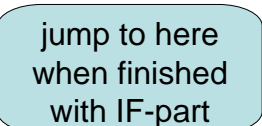
Som bytecode

Diagram illustrating the mapping of labels to specific bytes in the bytecode listing:

- Antall variable: Points to the first byte (00) of the first instruction.
- Antall prosedyrer: Points to the second byte (00) of the first instruction.
- Antall structer: Points to the third byte (00) of the first instruction.
- Lengden på variabelnavn: Points to the fourth byte (01) of the first instruction.
- Lengden på prosedyrenavn: Points to the fifth byte (02) of the first instruction.
- Main-nummer: Points to the sixth byte (00) of the first instruction.
- Ant: par, var, inst: Points to the seventh byte (00) of the first instruction.
- Variabeltype (Ref): Points to the eighth byte (00) of the first instruction.
- Struct nummer 0: Points to the ninth byte (00) of the first instruction.
- RETURN (24): Points to the 24th byte (00) of the first instruction.

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	ASCII
00000000:	00	00	00	01	00	02	00	01	00	00	00	0B	6D	79	47	6CmyGl
00000010:	6F	62	61	60	56	61	72	01	00	00	00	0E	00	16	00	04	obalVar.....
00000020:	00	00	00	00	00	01	4D	61	69	6E	00	18	00	04	00	02Main.....
00000030:	00	00	00	03	74	65	73	74	00	03	01	00	00	0C	00	00test.....
00000040:	03	00	64	16	00	00	07	00	02	43	6F	6D	70	6C	65		..d.....Comple
00000050:	78	03	03	00	00												x....

IF-statement

- Expression
- Instruction (NOP) 
- Statements (IF-part)
- Instruction (NOP) 
- Instruction (NOP) 
- Statments (ELSE-part)
- Instruction (NOP) 

Oppsummering

- Kontrollere statisk semantikk
 - Utvide syntakstre-nodene
 - Lage en context-klasse som håndterer symboltabell etc.
 - Kjøre testene i katalogen *test*
- Generere byte-kode
 - Utvide syntakstre-nodene
 - Holde orden på stack etc.
 - Kjøre RunMe.d
- Dokumentere løsningen i en rapport
- Frist: Fredag 8. mai

Kontaktinformasjon

- Send spørsmål når noe er uklart
- E-post: Andreas.Svendsen@sintef.no