Beskrivelse av programmeringsspråket Compila15

INF5110 - Kompilatorteknikk
Våren 2015

Her beskrives syntaksen og den statiske semantikken (hva som skal sjekkes av kompilatoren) til språket Compila15. Den dynamiske semantikken (altså hva som skal gjøres under utførelsen) skulle være rimelig opplagt, men eventuelle detaljer som er nødvendige vil vi komme tilbake til i forbindelse med Oblig 2.

1. Syntaks

I beskrivelsen av grammatikken under er ikke-terminaler skrevet med store bokstaver, og metasymbolene (i bokas betydning, altså de som brukes til å beskrive grammatikken) er :

-> , | , (,) , { , } , [,] , ”

Her betyr {...} gjentakelse null eller flere ganger, og [...] betyr at det inni kan være med eller ikke. Alt annet, som er skrevet som tette sekvenser, er terminalsymboler, og de med bare små bokstaver er reserverte (!) nøkkelord. Merk at alle terminalsymbolene er skrevet i anførselstegn for å skille dem fra de tilsvarende metasymbolene.

Det er noen spesielle terminaler som er skrevet med store bokstaver, og uten anførselstegn. Disse er betegnet *NAME, INT_LITERAL, FLOAT_LITERAL* og *STRING_LITERAL.*

- *NAME* skal starte med bokstav, og deretter være en sekvens av siffer, bokstaver og underscore. Store og små bokstaver regnes som forskjellige tegn. Alle nøkkelord skrives med små bokstaver, og de kan ikke brukes som vanlige navn.

- *INT_LITERAL* skal inneholde ett eller flere siffer.

- *FLOAT_LITERAL* skal inneholde ett eller flere siffer, fulgt av et punktum, fulgt av ett eller flere siffer.

- *STRING_LITERAL* skal bestå av en tekststreng innesluttet i anførselstegn (”). Strengen kan ikke inneholde linjeskift. Den semantiske ”verdien” av en *STRING_LITERAL* er kun det som er inni anførselstegn; selve anførselstegnene skal ikke inkluderes.

2. Datatyper

Språket har fire innebygde typer: “float”, “int”, “string” og “bool”. I tillegg utgjør hver klasse en type.

3. Klasser
Compila15 har en enkel form for klasser. Klasser kan kun inneholde variable og har ingen arv. Altså, ingen metoder, konstruktører eller subklasser. Klasser i Compila15 slekter ellers veldig på klasser i Java. Dermed kan variable av en klassetype enten peke på et objekt av klassen eller ha den spesielle verdien NULL.

4. Parametere
For presist å kunne snakke om parametere brukes følgende spesifiseringer i denne teksten:

• *Aktuell parameter* - Uttrykket eller variabelen gitt ved prosedyrekall.
• *Formell parameter* - En del av prosedyredefinisjonen.

Parametere i Compila15 kan overføres på to forskjellige måter:

Pass-by-value: Den formelle parameteren er en lokal variabel i prosedyren og verdien av den aktuelle parameteren vil bli kopiert til den formelle parameteren. For denne typen parameteroverføring angis ingen «modifikator» (i motsetning til for «pass-by-reference», se under).

Pass-by-reference: Den formelle parameteren vil her motta adressen til den aktuelle parameteren, som må være en variabel av riktig type. En tilordning til den formelle parameteren vil dermed forandre verdien til den aktuelle variabelen. Pass-by-reference markeres i Compila15 med nøkkelordet "ref", og dette må også brukes foran den tilsvarende aktuelle parameteren i et kall på prosedyren.

Et eksempel:

```
program
{
   proc swap(ref a : int, ref b : int){
      var tmp : int ;
      tmp := a ;
      a := b ;
      b := tmp ;
   }

   proc Main( ){
      var x : int;
      var y : int ;
      x := 42;
      y := 84;
      swap(ref x, ref y);
      // nå er x = 84, y = 42.
   }
}
```

5. Standardbibliotek
Programmet har et standardbibliotek med et sett av IO-prosedyrer.

- *proc int readint()* Leser en int fra standard inn.

- *proc float readfloat()* Leser en float fra standard inn.
- *proc int readchar()* Leser ett tegn fra standard inn og returnerer ASCII-verdien som en int. Returnerer -1 ved EOF.
- *proc string readstring()* Leser en string fra standard inn opp til første whitespace.
- *proc string readline()* Leser en tekstlinje fra standard inn.
- *proc printint(i:int)* Skriver en int til standard ut.
- *proc printfloat(f:float)* Skriver en float til standard ut.
- *proc printstr(s:string)* Skriver en string til standard ut.
- *proc printline(s:string)* Skriver en string til standard ut fulgt av et linjeskift.

6. Kommentarer

Kommentarer i Compila15 starter med // og fortsetter linjen ut (som i C++/Java).

7. Grammatikk

```
PROGRAM            -> "program" "{" { DECL }  "}"

DECL               -> VAR_DECL | PROC_DECL | CLASS_DECL

VAR_DECL           -> "var" NAME ":" TYPE ";"

PROC_DECL          -> "proc" NAME [ ":" TYPE ]
                      "(" [ PARAM_DECL { "," PARAM_DECL } ] ")"
                      "{" { DECL } { STMT } "}"

CLASS_DECL         -> "class" NAME "{" { VAR_DECL } "}"

PARAM_DECL         -> [ "ref" ] NAME ":" TYPE

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

VAR                -> NAME | EXP "." NAME

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

REL_OP             -> "<" | "<=" | ">" | ">=" | "=" | "<>"
```

```
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

IF_STMT             -> "if" EXP "then" "{" { STMT } "}"
                       [ "else" "{" { STMT } "}" ]
WHILE_STMT          -> "while" EXP "do" "{" { STMT } "}"

RETURN_STMT         -> "return" [ EXP ]

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

ACTUAL_PARAM        -> "ref" VAR | EXP

TYPE                -> "float" | "int" | "string" | "bool" | NAME
```

8. Presedens

Presedens-rekkefølge (fra lavest til høyest):

1. ||
2. &&
3. not
4. Alle relasjonsoperatorene
5. + og -
6. * og /
7. # (eksponensiering)
8. **.** (punktum, for tilgang til objektvariable)

9. Assosiativitet

- De binære operasjonene ||, &&, +, -, *, / og . er venstre-assosiative, mens # er høyre-assosiativ
- Relasjonoperatorene er ikke-assosiative (altså er f.eks. ”a<b+c<d”ulovlig).
- Det er altså lov å skrive ”not not not b”, og det betyr: ”not(not(not b))).

10. Semantikk (ikke viktig i Oblig 1)

Bruksforekomster av navn *uten* punktum foran bindes på vanlig måte til en deklarasjon: Let ut gjennom blokkene (altså prosedyrer eller program) som omslutter bruksstedet. Se på deklarasjonene i hver blokk, og velg den deklarasjonen der du først får treff. Om man ikke får treff er det en feil i programmet. Her regnes en parameter med til de lokale deklarasjonene i prosedyren.

Bruksforekomster av navn *med* punktum foran (altså: EXP "." NAME) bindes ved å se på typen til EXP (som må være av en klasse-type) og lete opp variabelen med navnet NAME i denne klassen. Om EXP ikke er av en klasse-type eller det ikke finnes noen variabel med navn NAME i klassen er programmet galt.

10.1 Uttrykk

- Det må sjekkes at uttrykk er typeriktig formet, på den opplagte måten. Hele uttrykket blir derved også tilordnet en type.
- Det må sjekkes at typen på begge sider av en tilordning er den samme.
  MERK: Det er lov både å lese og tilordne verdi til en formell parameter inne i prosedyren, og dette gjelder for både de som er overført "by ref" og andre (men tilordning vil ha svært forskjellig virkning i de to tilfellene)

- Det må sjekkes at typen av uttrykket etter "if" og etter "while" er "bool".

- Det må sjekkes at det uttrykket som kommer foran et punktum er av en klasse-type.

- Likeledes må det sjekkes at navnet som kommer etter et punktum er navnet på et attributt i (klasse-)typen til det foran punktumet.

10.2 Short-circuit evaluation

De logiske operatorene && og || benytter såkalt short-circuit evaluation. Det betyr at om verdien til det logiske uttrykket kan bestemmes etter å ha eksekvert den første delen, blir ikke den resterende delen eksekvert.

11. Typer og implisitt typekonvertering

Det er tillatt å tilordne uttrykk av typen int til variable av typen float. Det motsatte er ikke lov. Det finnes ingen operator for type-kast. Dersom en aritmetisk operasjon har minst en operand av typen float skal operasjonen evalueres med flyttallsaritmetikk. Svaret vil da ha typen float. Eksponensiering gjøres alltid med flyttallsaritmetikk, og svaret har typen float.

12. Prosedyrer

- I en prosedyre skal alle deklarasjonene komme før eksekverbare setninger (statements). Man kan her bruke de samme type-deklarasjoner som man kan på ytterste program-nivå, altså variable, prosedyrer og klasser.

- Prosedyrer kalt i et uttrykk må ha en definert returtype, og denne må også passe inn i uttrykket.

- Antall og typer på parametere må stemme overens mellom kallsted og deklarasjon, også m.h.t. overføringsmåte ("by ref" eller ikke).

- Return-setninger kan bare forekomme inne i prosedyre-deklarasjoner, og de angir at prosedyren skal terminere, og eventuelt levere en verdi tilbake.

- Dersom en prosedyre er deklarert uten returtype behøver ikke prosedyren ha noen return-setning i det hele tatt. Om utførelsen går ut gjennom siste setning vil prosedyren returnere på vanlig måte, som om det stod en return-setning bakerst.

- Dersom en prosedyre har en returtype må det være en return-setningen (med et uttrykk) som siste setning i prosedyren.

13. Generelt

- Det må ikke være dobbeltdeklarasjoner innen en blokk (og her regnes samme navn på prosedyre, klasse eller variabel som en dobbeltdeklarasjon).

- Navnet på en formell parameter må ikke kollidere med navn på lokale deklarasjoner inne i prosedyren.

- Det må sjekkes at alle navn som brukes er deklarerte.

- Ethvert program må ha en prosedyre som heter Main, og det er denne som kalles når programmet starter.