Beskrivelse av programmeringsspråket Simpila

INF5110 - Kompilatorteknikk
Våren 2012

Her beskrives syntaksen og den statiske semantikken (hva som skal sjekkes av kompilatoren) til språket Simpila. 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 kan være med eller ikke.

Alt annet, som er skrevet som tette sekvenser, er terminalsymboler, og de med 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 beståav en tekststreng innesluttet i anførselstegn (”). Strengen kan ikke inneholde linjeskift. Den semantiske ”verdien” av en 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

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

4. Parametere

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

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

Parametere i Simpila kan enten overføres ved *pass-by-value* eller *pass-by-reference*.

Pass-by-value vil kopiere verdien av den aktuelle parameteren til den formelle parameteren.

Pass-by-reference vil gi den formelle parameter en implisitt referanse til den aktuelle parameter, som må være en variabel av riktig type. Derfor vil en tilordning til den formelle parameteren forandre verdien til den aktuelle parameter. Pass-by-reference markeres i Simpila med nøkkelordet "ref", som må brukes både i prosedyredefinisjonen og i prosedyrekallet.

Et eksempel:

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

proc Main( ){
   var int x;
   var int y;
   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(int i)* Skriver en int til standard ut.
- *proc printfloat(float f)* Skriver en float til standard ut.
- *proc printstr(string s)* Skriver en string til standard ut.
- *proc printline(string s)* Skriver en string til standard ut fulgt av et linjeskift.

6. Kommentarer

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

7. Grammatikk

```
PROGRAM            -> DECL { DECL }

DECL               -> VAR_DECL | PROC_DECL | CLASS_DECL

VAR_DECL           -> "var" TYPE NAME ";"

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

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

PARAM_DECL         -> [ "ref" ] TYPE NAME

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 objektvariabler)

9. Assosiativitet

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

10. Semantikk (ikke viktig i oblig 1)

Bruksforskomster 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.

Alle navn må være deklarert tekstlig før de brukes.

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.

- 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 evaluerer til et objekt.
- Likeledes må det sjekkes at navnet som kommer etter et punktum er navnet på et attributt av klassetypen 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 cast-operator. 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). Disse deklarasjonene kan inkludere alt som kan deklareres på ytterste nivå, altså variable, prosedyrer og klasser.

- Returverdien til en prosedyre kan kun være typer som er synlige i det skopet som prosedyre selv er definert i.

- Prosedyrer brukt i uttrykk må ha en definert returtype.

- Det må sjekkes at antall og typer på parametere stemmer overens mellom kallsted og deklarasjon.

- Return-setninger kan bare forekomme inne i prosedyre-deklarasjoner, og de angir at prosedyren skal terminere.

- Dersom prosedyren er deklarert uten returtype skal return-setningen ikke ha noe uttrykk, ellers skal den ha et uttrykk av prosedyrens type, og det skal enten være "ref" på begge eller ingen.

- Dersom prosedyren er deklarert med en retur-type må det sjekkes at prosedyren inneholder minst ett return-statement som vil nås uansett.

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 prosedyrens lokale deklarasjonsnavn.

- 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.

- Det må sjekkes at nøkkelordet “ref” på en parameter brukes både ved kall og i prosedyresignaturen, eller ingen av stedene.