Progetto del Corso di Laboratorio di Linguaggi

Anno Accademico 2022/2023

Linguaggio HaveFun

Assegnamento

L'obiettivo del progetto è l'implementazione del nuovo linguaggio HaveFun. Per fare questo, si estenda l'interprete Imp con le seguenti nuove definizioni:

■ **Struttura dei Programmi.** Un programma HaveFun avrà la seguente struttura:

$$fun_1 \ldots fun_n\ com$$

ovvero,

1- una sequenza arbitrariamente lunga e potenzialmente vuota di definizioni di funzione (descritte al punto successivo);
2- seguite da un singolo comando (non opzionale).

Fondamentalmente, l'unica differenza strutturale tra i programmi scritti in Imp e HaveFun è la possibilità del secondo di definire una serie di funzioni all'inizio dei programmi.

■ **Definizione di Funzione.** Una definizione di funzione ha la seguente sintassi:

$$fun\ id_0\ (id_1, \ldots, id_n)\ \{\ com; return\ exp\ \}$$

Si presti attenzione ai seguenti punti:

- sono ammesse funzioni con zero parametri, ad esempio: *fun f() { return 1 }*
- il corpo (ovvero, *com*) della funzione è opzionale (vedi esempio precedente);
- se il corpo della funzione è presente, allora dovrà essere obbligatoriamente seguito da un punto e virgola;
- il *return* deve comparire obbligatoriamente e solo come ultimo statement durante la definizione di una funzione;
- non sono ammesse funzioni che non ritornano nessun valore, ad esempio

  fun f() { x = 0; y = 1; return }

  non sarà un programma valido. In altre parole, *exp* non è opzionale.

L'interpretazione di una dichiarazione di funzione ha l'obiettivo di creare un bind tra il nome logico della funzione id_0 e la sua implementazione, in modo tale che sia poi richiamabile dal codice

(vedi il punto successivo). Per fare questo, dovrete modificare/estendere in modo opportuno la configurazione dell'interprete in modo che tenga traccia di tutte queste definizioni.

Hint: Implementare una classe *FunValue* estendendo Value che memorizza i seguenti aspetti: nome della funzione, lista dei nomi dei parametri, corpo della funzione (ovvero, il *ComContext*), ed espressione di ritorno (*ExpContext*).

Inoltre, la visita di una definizione di funzione dovrà assicurarsi che

- la funzione id_0 **non sia già definita**. In questo caso, stampate a video un errore ed interrompete l'esecuzione. Ad esempio:

  ```
  fun f(x, y, z) {
      return x
  }

  fun f(x, y, z) {
      return y
  }
  ```

  stamperà a video

  ```
  Fun f already defined
  @5:0
  ```

- i parametri formali della stessa funzione (ovvero, $id_1, \ldots, id_n$) devono essere tutti diversi tra loro. Ad esempio:

  ```
  fun f(x, y, x) {
      return x
  }
  ```

  stamperà a video

  ```
  Parameter name x clashes with previous parameters
  @1:0
  ```

■ **Chiamata di Funzione.** La chiamata di una funzione sarà un'espressione con la seguente sintassi:

$$id\ (exp_1, \ldots, exp_n)$$

In fase di visiting, dovrete prestare attenzione ai seguenti aspetti:

- assicurarsi che la funzione *id* **sia stata definita**, in caso contrario ritornare un errore. Ad esempio:

  ```
  fun f(x, y, z) {
      return x
  }

  out(g(4))
  ```

  stamperà

  ```
  Function g used but never declared
  @5:4
  ```

- assicurarsi che il numero di argomenti con cui la funzione viene chiamata **corrisponda** al numero di parametri formali con cui è stata definita. Ad esempio

  ```
  fun f(x, y, z) {
      return x
  }

  out(f(4))
  ```

  stamperà

  ```
  Function f called with the wrong number of arguments
  @5:4
  ```

  Un aspetto molto importante riguarda le **regole di scoping**. Ogni funzione chiamata vede nel suo scope i suoi parametri formali ed eventualmente **le variabili globali** (vedere la sezione successiva). Le funzioni non hanno modo nè di accedere nè di modificare sia le variabili non globali esterne alle funzioni che le variabili dichiarate in altre funzioni. Ovviamente il corpo della funzione chiamata può definire nuove variabili utilizzabili all'interno della funzione stessa. Esempi:

- Programma:

  ```
  fun f(x) {
      return y
  }
  ```

```
y = 5;
out(f(y))
```

Output:

```
Variable y used but never instantiated
@2:9
```

- Programma:

```
fun f(x) {
    return y.g
}

global y = 5;
y = 3;
out(f(y.g))
```

Output:

```
5
```

- Programma:

```
fun f(x) {
    y = 0;
    out(y);
return y
}

y = 5;
out(f(y));
out(y)
```

Output:

```
0
0
5
```

- Programma:

```
fun f(x) {
    y = 0;
    out(y);
    out(y.g);
    return y
}

global y = 3;
y = 5;
out(f(y));
out(y)
```

Output:

```
0
3
0
5
```

- Programma:

```
fun f(x) {
    y = 0;
    r = g(y);
   out(y);
    out(r);
    return y
}

fun g(x) {
    y = 10;
    out(y);
    return y
}

y = 5;
out(f(y));
out(y)
```

Output:

10
0
10
0
5

■ Variabili globali.

Si differenziano per tre peculiarità:

1. Necessitano della keyword "global" nella dichiarazione, esempio:

$$\ldots \textit{global } x = 50; \ldots$$

2. Le funzioni hanno modo di accedere ad esse e di modificarne il valore.
3. In caso di utilizzo di una variabile globale è necessario aggiungere "*.g* " all'identificatore della variabile, esempio:

$$\ldots x.g + 10; \ldots$$

NB: possono esistere variabili locali aventi lo stesso nome di una qualunque variabile esterna alla funzione considerata.

■ Non determinismo.

Aggiungere un comando "*nd* " che permetta la scelta non deterministica di due comandi, con la seguente sintassi:

$$\{\ com_1\ \}\ nd\ \{\ com_2\ \}$$

Esempi:

- Programma:

```
fun f(x) {
    y = x + 10;
return y
}

y = 5;

{ out(f(y)) } nd { out(y) }
```

Possibili Output:

15

Oppure

5

- Programma:

```
fun f(x) {
    y = x + 10;
return y
}

global y = 5;
y = 5;

{ out(f(y)) } nd { { out(y) } nd { out(y.g + 10) } }
```

Possibili Output:

15

Oppure

5

■ ArnoldC.

Aggiungere la possibilità di inserire programmi scritti in ArnoldC. Possono essere inseriti sia nel corpo delle funzioni, che nel corpo principale del programma sfruttando questo "costrutto":

$\{ "ArnoldC-Programs" \}$

Viene richiesta l'implementazione dei seguenti costrutti:

- *IT'S SHOWTIME --- YOU HAVE BEEN TERMINATED*
- *TALK TO THE HAND*
- *HEY CHRISTMAS TREE --- YOU SET US UP*
- *GET TO THE CHOPPER --- HERE IS MY INVITATION --- ENOUGH TALK*
- *GET UP*
- *GET DOWN*
- *YOU'RE FIRED*

- *HE HAD TO SPLIT*
- *YOU ARE NOT YOU YOU ARE ME*
- *LET OFF SOME STEAM BENNET*
- *CONSIDER THAT A DIVORCE*
- *KNOCK KNOCK*
- *@I LIED --- @NO PROBLEMO*
- *BECAUSE I'M GOING TO SAY PLEASE --- BULLSHIT --- YOU HAVE NO RESPECT FOR LOGIC*
- *STICK AROUND --- CHILL*

Solo per i progetti svolti in coppia:

- ***LISTEN TO ME VERY CAREFULLY***
- ***HASTA LA VISTA, BABY***
- ***I NEED YOUR CLOTHES YOUR BOOTS AND YOUR MOTORCYCLE***
- ***GIVE THESE PEOPLE AIR***
- ***I'LL BE BACK***
- ***GET YOUR ASS TO MARS***
- ***DO IT NOW***
- **Un nuovo costrutto per permettere di utilizzare/modificare il valore di una variabile globale all'interno dei programmi "ArnoldC":**

ARNOLD OR SLY x.g y

Le modifiche apportate alla variabile y si riflettono sulla variabile globale x.

Per la semantica dei singoli costrutti, si consiglia di consultare https://github.com/lhartikk/ArnoldC/wiki/ArnoldC .

Esempi:

- Programma:

```
fun f() {
    $${
    IT'S SHOWTIME
        TALK TO THE HAND "Anna Karenina"
    YOU HAVE BEEN TERMINATED
    }$;
    return 42
}

out(f())
```

Output:

```
Anna Karenina
42
```

- Programma:

```
fun f() {
    $${
    IT'S SHOWTIME
        HEY CHRISTMAS TREE levin
            YOU SET US UP @NO PROBLEMO
        HEY CHRISTMAS TREE n
            YOU SET US UP 0.0
        STICK AROUND levin
            GET TO THE CHOPPER n
              HERE IS MY INVITATION n
              GET UP 42.0
            ENOUGH TALK
            TALK TO THE HAND n
            GET TO THE CHOPPER levin
              HERE IS MY INVITATION 420.0
              LET OFF SOME STEAM BENNET n
            ENOUGH TALK
        CHILL
    YOU HAVE BEEN TERMINATED
    }$;
    return 42
}

out(f())
```

Output:

```
42.0
84.0
126.0
168.0
210.0
```

252.0
294.0
336.0
378.0
420.0
42

Linee Guida

- Creare un nuovo progetto di nome havefun a partire dal codice (opportunamente modificato) contenuto in imp.zip.
- JDK: 17 (evitare di utilizzare versioni differenti).
- ANTLR: 4.12.0 /4.11.1 (https://mvnrepository.com/artifact/org.antlr/antlr4).
- Ambiente test: Debian 10.
- Consegna: su moodle sarà possibile caricare un archivio nome_cognome.zip/.tar.gz (per i progetti svolti in coppia (nome1_cognome1_nome2_cognome2.zip/.tar.gz) contenente **tutti** i file del progetto (IntelliJ).
- Testing: per verificare la correttezza dell'implementazione, è possibile effettuare una serie di test rispetto a programmi corretti/non corretti. **IL PROGETTO CONSEGNATO DEVE PASSARE TUTTI I TEST.**

In caso di dubbi, perplessità,… potete contattarmi via e-mail: marco.vedovato@studenti.univr.it