Università di Bologna

Corso di Compilatori e Interpreti

Dip. di Informatica

Progetto SimpLanPlus

Benetton Alessandro [0001038887]

Crespan Lorenzo [0001038888]

Zhiguang Li [0001029938]

23 aprile 2022

Indice

1 Descrizione progetto

1.1 Definizione di SimpLanPlus

SimpLanPlus è un linguaggio imperativo avente le seguente caratteristiche:

- I parametri delle funzioni possono essere passati per valore o per riferimento:
  - In caso di **passaggio per valore**, alla funzione viene passata una copia degli argomenti e le potenziali modifiche non hanno ripercussioni sul valore originale delle variabili passate alla funzione;
  - In caso di **passaggio per riferimento** (utilizzando il prefisso "var" sul parametro), alla funzione viene passato un puntatore all'indirizzo della variabile. Ogni modifica avrà ripercussioni sul valore originale della variabile passata alla funzione;
- Le funzioni non sono annidate, ergo una funzione non può essere definita nel corpo di un'altra;
- Ammette la ricorsione, ma non la mutua ricorsione (detta anche ricorsione indiretta). Quest'ultima si verifica quando nel codice di una funzione viene richiamata un'altra funzione che a sua volta richiama la prima (vedi 1.1).

```
public class MyRecursiveMethods {

    // funzione ping richiamata pong
    public static int ping(int n) {
        if (n < 1) return 1;
        else return pong(n - 1);
    }

    // funzione pong richiamata ping
    public static int pong(int n) {
        if (n < 0) return 0;
        else return ping(n/2);
    }
}
```

Listing 1: Esempio di mutua ricorsione

1.2 Obiettivi progetto

Nel progetto completo è richiesto lo sviluppo di un compilatore per SimpLanPlus e la produzione di un report, nel quale devono essere illustrate le scelte progettuali.

2 Esercizio 1

2.1 Obiettivo

Viene richiesto dall'analizzatore lessicale di riportare gli errori in un file di output.
Di seguito sono riportati tre codici di esempio e i relativi output di errore.

2.2 Esempi errori segnalati

Nei seguenti codici è stata verificata la capacità dell'analizzatore lessicale di identificare token non presenti all'interno della grammatica. Questa tipologia di errore è causata principalmente da errori di battitura.

Nel codice sono presenti anche errori di natura semantica che non devono essere riportati dall'analizzatore lessicale.

2.2.1 Esempio 1

```
1     {
2         7
3         bool isEven (int value){
4             if( value == 0){
5                 retun true;
6             } else {
7                 if (value === 1){
8                     return false;
9                 } else {
10                    return isEvens(value - 2);
11                }
12            }
13        }
14
15        bool num = 10a;
16        print (isEven(num));
17    }
```

Funzione ricorsiva per la valutazione di valore pari o dispari.

Errori riportati all'interno del documento "Test_1.simplan_[Data].log" generato dall'analizzatore lessicale:

```
1     Errors:
2       [ERROR] Error at pos 2:4. extraneous input '7' expecting {'void',
      'if', 'return', 'print', '{', '}', 'int', 'bool', ID}
3       [ERROR] Error at pos 5:18. no viable alternative at input '
      retuntrue'
4       [ERROR] Error at pos 7:24. extraneous input '=' expecting {'(',
      '!', '-', BOOL, ID, NUMBER}
5       [ERROR] Error at pos 15:17. extraneous input 'a' expecting ';'
```

Errori riportati da SimpLanPlus sul codice riportato precedentemente

Il primo errore, dovuto al carattere "7" inserito in riga 2, simula una pressione accidentale di un tasto da parte dell'utente mentre si muove nel codice.

Il secondo e il quarto errore, rispettivamente in riga 5 e 10, simulano un errore di battitura in fase di stesura del codice.

Il terzo errore, in riga 7, simula un utente abituato a programmare in un differente linguaggio che utilizza operatori differenti.

2.2.2 Esempio 2

```
1       {
2       int fibonacciSequence (int value){
3           if (value !> 1)[
4               return value:
5           ]
6           return fib(value-1) + fib(value-2);
7
8
9       int num = 10;
10      print (fibonaciSequence(num));
11      }
```

Funzione ricorsiva per la valutazione dell'ennesimo numero della sequenza di Fibonacci.

Errore riportato all'interno del documento "Test_2.simplan_[Data].log" in output:

```
1       Errors:
2         [ERROR] Error at pos 3:18. mismatched input '!' expecting {')',
        '*', '/', '+', '-', '<', '>', '<=', '>=', '==', NEQ, '&&', '||'}
3         [ERROR] Error at pos 3:23. token recognition error at: '['
4         [ERROR] Error at pos 4:24. token recognition error at: ':'
5         [ERROR] Error at pos 5:8. token recognition error at: ']'
6         [ERROR] Error at pos 6:8. missing ';' at 'return'
7         [ERROR] Error at pos 9:4. extraneous input 'int' expecting {'if',
       'return', 'print', '{', '}', ID}
8         [ERROR] Error at pos 9:16. token recognition error at: ';'
9         [ERROR] Error at pos 10:4. missing ';' at 'print'
```

Errori riportati da SimpLanPlus sul codice riportato precedentemente

Gli errori in riga 3, 4, e 5 sono già stati analizzati nella sezione precedente.

Gli errori alla riga 6 e 10 sono dovuti ad errori nelle righe precedenti.

L'errore alla riga 9 simula un utente che copia una riga di codice da una sorgente esterna che utilizza una codifica differente. Il carattere ";" presente alla fine della riga 9, anche se uguale dal punto di vista grafico, ha una codifica numerica differente rispetto a quello standard.

2.2.3 Esempio 3

```
1     {
2         int addNumbers(int n) {
3             if (9 !== 0) return n + addNumbers(n - 1);
4             else return n;
5         }
6
7         int num1 = 10;
8         int result == addNumbers(num1);
9         printf(result);
10
11        int num2 = 10;
12        result = addNumbers(num2);
13        printf(result);
14    }
```

Funzione ricorsiva per la stampa della somma dei primi n numeri.

Errore riportato all'interno del documento "Test_3.simplan_[Data].log" in output:

```
1     Errors:
2       [ERROR] Error at pos 3:14. extraneous input '=' expecting {'(',
      '!', '-', BOOL, ID, NUMBER}
3       [ERROR] Error at pos 8:13. no viable alternative at input '
      intresult=='
4       [ERROR] Error at pos 11:2. extraneous input 'int' expecting {'if
      ', 'return', 'print', '{', '}', ID}
5
```

Errori codice test

L'errore alla riga 3 è già stato visto alla sezione precedente.

L'errore alla riga 8 è dovuto ad un carattere presente nel lexer, ma che non fa parte della sintassi di dichiarazione e assegnamento.

L'errore alla riga 11 è dovuto ad una dichiarazione di variabile in seguito ad uno statement. Questo non è consentito poichè, per definizione del linguaggio, devono essere definite prima tutte le dichiarazioni e poi tutti gli statement.

3 Esercizio 2

3.1 Obiettivo

Viene richiesto di implementare nel linguaggio una tabella dei simboli sfruttando una lista di hash-table o un hash-table di liste.

Al termine dell'implementazione, SimpLanPlus deve essere in grado di verficare e riportare errori semantici come:

- la presenza di variabili/funzioni non dichiarate;
- la presenza di variabili/funzioni dichiarate più volte nello stesso ambiente.

3.2 Tabella dei simboli

La Symbol Table è stata implementata come una Hash Table di liste (più di preciso Stack).

Considerando che le due soluzioni sono equivalenti e che entrambe hanno dei vantaggi e degli svantaggi, la scelta è stata fatta valutando il numero di aggiunte e rimozioni stimato nel corso del programma.

Dato che il numero di aggiunte e ricerche nella symbol table è generalmente superiore al numero di rimozioni, e dato che nel caso di HashMap di ArrayList la complessità di aggiunte e ricerche è O(1) abbiamo ritenuto ottimale implementare questa versione invece che quella presente nell'esempio fornitoci (ArrayList di HashMap).

3.2.1 Funzionamento

La classe "SymbolTableWrapper" contiene l'effettiva SymbolTable rappresentata come "HashMap<String, ArrayDeque<STentry»".

All'interno della classe sono presenti tre funzioni utili alla gestione della SymbolTable. Il primo metodo serve per aggiungere un STentry alla SymbolTable, verificando che non sia stato aggiunto in precedenza allo stesso livello. Il secondo metodo ricerca all'interno della SymbolTable la prima STentry avente ID uguale a quello specificato. L'ultimo metodo consente la rimozione di tutte le STentry ad un determinato livello di nesting (questa funzione rappresenta la più costosa computazionalmente, con un costo pari a O(n), dove n indica il numero di entry nella Hash Table).