Relazione Progetto Compilatori
SimpLanPlus
Parte I

Stefano Notari
Francesco Santilli

12 Aprile 2022

Indice

1 Esercizio 1

Per intercettare gli errori sintattici abbiamo creato una classe che estende il listener "BaseErrorListener", che si occupa di aggiungere in una lista i messaggi di errore per poi memorizzarli nel file di log. La classe così creata viene aggiunta al parser, se sono presenti errori non procede con la fase successiva. I log sono consultabili nella cartella **src/output/error_logger_TS.log** (dove TS indica il timestamp).

1.1 Esempio 1

In questo esempio abbiamo preso uno tra i codici proposti per la verfica dell'esercizio 3 e lo abbiamo leggermente modificato in modo da ottenere alcuni errori lessicali.
Vengono inizializzati 3 interi e in base al valore di c vengono effettuati degli assegnamenti.

```
{
    int a
    int b
    int c = 1 ;
    if (c >>> 1) {
        b == c ;
    } else {
        a = b
    }
}
```

```
====================LEXER ERRORS=========================
Number of errors: 5
  line 3: no viable alternative at input 'intaint'
  line 5: mismatched input '>' expecting {'(', '-', '!', BOOL
    , ID, NUMBER}
  line 5: extraneous input '>' expecting {'(', '-', '!', BOOL
    , ID, NUMBER}
  line 6: no viable alternative at input 'b=='
  line 9: missing ';' at '}'
====================LEXER ERRORS=========================
```

1.2 Esempio 2

In questo esempio il programma calcola il fattoriale di un numero dato in input. Gli errori lessicali aggiunti al programma sono i seguenti:

- il parametro formale della funzione non contiene la dichiarazione del suo tipo
- errore di scrittura del return
- aggiunto un carattere nell'ultimo return

```
{
    int num = 9;
    int fatt(num) {
        if(num == 0) {
            retur 1;
        }
        return num * fatt num - 1);
    }
    print(fatt(num));
}
```

```
====================LEXER ERRORS=========================
Number of errors: 3
  line 3: extraneous input 'num' expecting {')', 'int', 'bool
    ', 'var'}
  line 5: no viable alternative at input 'retur1'
  line 7: missing ';' at 'num'
====================LEXER ERRORS=========================
```

1.3 Esempio 3

In questo terzo esempio abbiamo scritto un frammento di codice al quale abbiamo aggiunto alcuni errori lessicali, in particolare sono stati inseriti dei caratteri speciali che il programma non si aspetta.

```
{
    int a(int b) {
        return b
        };
    int b;
    int c = 1 ;
    )
```

```
8       if (b =<= 10!) {
9           b = c
10      } els!e {
11          a != b ;
12      }
13  }
```

```
1   ====================LEXER ERRORS=========================
2   Number of errors: 7
3     line 3: extraneous input '}' expecting ';'
4     line 4: extraneous input 'int' expecting {'{', '}', 'print
        ', 'return', 'if', ID}
5     line 6: extraneous input ')' expecting {'{', '}', 'void', '
        int', 'bool', 'print', 'return', 'if', ID}
6     line 7: mismatched input '=' expecting {')', '-', '*', '/',
        '+', '<', '<=', '>', '>=', '==', '!=', '&&', '||'}
7     line 9: missing ';' at '}'
8     line 9: no viable alternative at input 'els!'
9     line 10: no viable alternative at input 'a!='
10  ====================LEXER ERRORS=========================
```

2 Esercizio 2

La SymbolTable è stata implementata tramite una lista di HashTable così definita:

- key: ID della variabile funzione
- value: sia per le funzioni che per le variabili memorizziamo il tipo, in più per le funzioni si memorizzano la lista di parametri formali con i relativi tipi (necessario per la fase successiva)

Il motivo di dell'implementazione attraverso una lista di HashTable è dovuta al fatto di utilizzare il meccanismo di Listening che ci permette di avere una notifica di entrata in un nuovo blocco e dell'uscita.

2.1 Esempio

Per questa fase abbiamo scritto un breve programma per evidenziare gli errori riguardanti l'utilizzo di variabili/funzioni non dichiarate o la molteplice definizione nello stesso blocco.

```
{
    int x;
    int y;
    int y;
    int z = 14;
    int fun1(bool b) {
        bool b = false;
        return 0;
    }
    int fun1(){
        return 15;
    }
    int fun2(int a, bool b){
        a = 17;
        return 15;
    }
    if (z > 1) {
       int x;
       k();
       y = g;
    } else {
       x = z;
    }
}
```

```
======================SEMANTIC ERRORS=====================
Number of errors: 5
  line 4: Multiple declaration of variable y
  line 7: Multiple declaration of variable b
  line 10: Multiple declaration of function fun1
  line 19: Function k not declared
  line 20: Variable g not declared
======================SEMANTIC ERRORS=====================
```