Compilatori 2020/2021

Specifiche compilatore toy

Edoardo carpentiero – attilio della greca

Sommario

[1 Analisi Lessicale 2](#_Toc62996997)

[2 Analisi Sintattica 2](#_Toc62996998)

[3 Analisi Semantica 2](#_Toc62996999)

[3.1 Gestione scoping 2](#_Toc62997000)

[3.2 Type Checking 2](#_Toc62997001)

[3.2.1 Type System 2](#_Toc62997002)

[4 Gestione degli errori 3](#_Toc62997003)

[5 Generarione codice Intermedio (Codice C) 3](#_Toc62997004)

# Analisi Lessicale

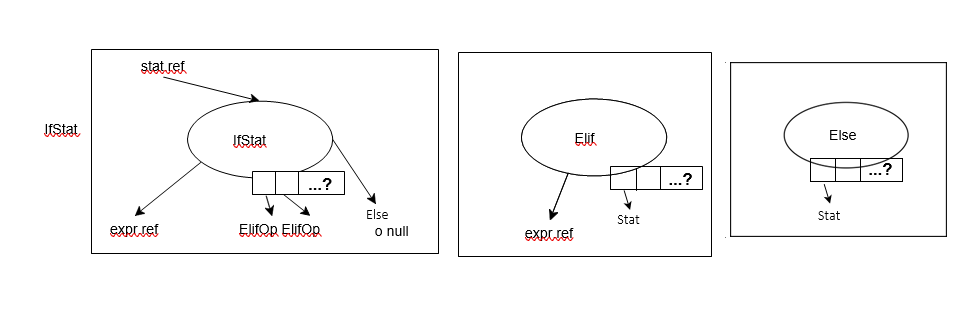
|  |  |
| --- | --- |
| Token | Pattern |
| SEMI | ; |
| COMMA | , |
| ID | Any sequence of letters, digits and underscores,starting with a letter or an underscore |
| INT | int |
| STRING | string |
| FLOAT | float |
| BOOL | bool |
| LPAR | ( |
| RPAR | ) |
| COLON | : |
| PROC | proc |
| CORP | corp |
| VOID | voit |
| IF | if |
| THEN | then |
| ELIF | elif |
| FI | fi |
| ELSE | else |
| WHILE | while |
| DO | do |
| OD | od |
| READ | realn |
| WRITE | write |
| ASSIGN | := |
| PLUS | + |
| MINUS | - |
| TIMES | \* |
| DIV | / |
| EQ | = |
| NE | <> |
| LT | < |
| LE | <= |
| GT | > |
| GE | = |
| AND | && |
| OR | or |
| NOT | ! |
| NULL | null |
| TRUE | true |
| FALSE | false |
| INT\_CONST | any integer number |
| FLOAT\_CONST | any floating point number |
| STRING\_CONST | any string between " |

# Analisi Sintattica











# Analisi Semantica

## Gestione scoping

La gestione dello scoping avviene attraverso due visite dell'albero sintattico

* Il visitor ***CreateScopingVisitor.java*** consentirà di individuare i livelli di scope presenti nel codice sorgente. Tale attività include il popolamento della tabella dei simboli principale (Program) inserendo gli identificativi delle procedure e i rispettivi tipi dei parametri e tipi di ritorno
* In ***ScopeVisitor.java*** viene ultimato il popolamento delle tabelle dei simboli associate ai vari livelli di scope, inserendo le variabili e i controlli necessari.

## Type Checking

Il type checking avviene attraverso una sola visita dell'AST

Il visitor ***TypeCheckingVisitor.java*** effettuerà il type checking. Quest'ultimo avviene mediante l'utilizzo di *OperationTable.java* e delle regole fornite dal *TypeSystem*.

### Type System

* costrutto **Proc**, nodo *Proc*

**IF** il nodo BodyProc non presenta errori di tipo

**THEN** Proc non ha errori di tipo

**ELSE** typeError

* costrutto **Proc**, nodo *BodyProc*

**IF** il nodo VarDecl non ha errori di tipo **AND** il nodo Statement non ha errori di tipo **AND** il nodo ReturnExp non ha errori di tipo

**THEN** BodyProc non ha errori di tipo

**ELSE** typeError

* costrutto **Proc**, nodo *ReturnExpr*

**IF** il numero dei nodi figli coincide con il numero dei nodi figli di ResultType **AND** i nodi figli hanno lo stesso tipo di quelli dichiarati rispettivamente in ResultType

**THEN** ReturnExpr non ha errori di tipo

**ELSE** typeError

* costrutto **Proc**, nodo *ReturnExpr*

**IF** non ci sono nodi figli **AND** in ResultType esiste solo il tipo VOID

**THEN** ReturnExpr non ha errori di tipo

**ELSE** typeError

* costrutto **CallProc**, nodo *CallProc*

**IF** il numero dei nodi del secondo figlio coincide con il numero dei nodi figli di ParDecl **AND** i nodi figli hanno tipi compatibili([[1]](#footnote-1)) di quelli presenti in ParDecl

**THEN** CallProc non ha errori di tipo

**ELSE** typeError

* costrutto **write**, nodo *WriteStatement*

**IF** i nodi figli non hanno errori di tipo

**THEN** WriteStatement non ha errori di tipo

**ELSE** typeError

Nella dichiarazione delle variabili, se una variabile viene inizializzata con una procedura avente 0 o piu di un resulttype viene lanciato un errore semantico.

L'istruzione Assignment accetta il cast implicito da FLOAT a INT e viceversa.

|  |  |  |  |
| --- | --- | --- | --- |
| **Op2** | **FirstOperand** | **SecondOperand** | **Result** |
| + - \* / | Integer | Integer | Integer |
| + - \* / | Integer | Float | Float |
| + - \* / | Float | Integer | Float |
| + - \* / | Float | Float | Float |
| And Or | Bool | Bool | Bool |
| <, <=, >, >=, = | Bool | Bool | Bool |
| <, <=, >, >=, = | Integer | Integer | Bool |
| <, <=, >, >=, = | Integer | Float | Bool |
| <, <=, >, >=, = | Float | Float | Bool |
| <, <=, >, >=, = | Float | Integer | Bool |
| <,>,= | String | String | Bool |

|  |  |  |
| --- | --- | --- |
| Op1 | Operand | Result |
| - | Integer | Integer |
| - | Float | Float |
| not | Bool | Bool |

***Tabella Op1***

***Tabella Op2***

|  |  |
| --- | --- |
| ***While Statment*** |  |
| ***If Statement*** |  |
| ***Elif Statement*** |  |
| ***Else Statement*** |  |
| ***Return expr*** |  |
| ***CallProc*** |  |

# Gestione degli errori

* Durante la fase di riconoscimento dei token, se quest'ultimo presenta un errore, è stato sovrascritto il metodo *scan()* della classe **Parser.java** (generata da CUP) in modo da generare correttamente il tipo di errore che si è verificato durante l'analisi lessicale.
* Nell'analisi sintattica, gli errori vengono gestiti da CUP e mostrati mediante la sovrascrittura del metodo *report\_error()*. Tale metodo, una volta chiamato, consente la visualizzazione del tipo di errore riscontrato e i relativi dettagli.
* Se tra le fasi dell'analisi semantica (Scoping e Type checking) dovessero verificarsi degli errori, la fase successiva non verrà eseguita.

# Generarione codice Intermedio (Codice C)

Per consentire la traduzione delle procedure aventi piu tipi/valori di ritorno sono state utilizzate le *struct.*

1. ammesso il cast implicito da FLOAT a INT e viceversa [↑](#footnote-ref-1)