Progetto Compilatori

A.A. 2021/2022 Gennaro Pio Rimoli 0522501296

[Scelte Progettuali 1](#_Toc93438215)

[***Analizzatore Lessicale – JFlex*** 1](#_Toc93438216)

[***Analizzatore Sintattico – CUP*** 1](#_Toc93438217)

[***Generazione dell’Abstract Syntax Tree*** 1](#_Toc93438218)

[***Analizzatore Semantico*** 2](#_Toc93438219)

[***Generazione del codice intermedio*** 2](#_Toc93438220)

[Regole di Type Checking 3](#_Toc93438221)

[Tipi Primitivi 3](#_Toc93438222)

[Dichiarazioni di Variabili 3](#_Toc93438223)

[Operazioni Unarie 3](#_Toc93438224)

[Operazioni Binarie 3](#_Toc93438225)

[Chiamata a Funzione senza ritorno 3](#_Toc93438226)

[Chiamata a Funzione con ritorno 3](#_Toc93438227)

[Statement 3](#_Toc93438228)

[If-Then 3](#_Toc93438229)

[If-Then-else 3](#_Toc93438230)

[While 3](#_Toc93438231)

[Read 4](#_Toc93438232)

[Read con Print 4](#_Toc93438233)

[Print 4](#_Toc93438234)

[Assign 4](#_Toc93438235)

[Dichiarazione 4](#_Toc93438236)

[Lista di Istruzioni 4](#_Toc93438237)

[Tabelle delle operazioni 4](#_Toc93438238)

# Scelte Progettuali

## ***Analizzatore Lessicale – JFlex***

È stata gestita la posizione inziale in cui vengono generati i seguenti errori:

1. Commento non chiuso correttamente
2. Stringa non chiusa correttamente
3. Carattere non riconosciuto

Nei primi due casi, al passaggio in uno dei seguenti stati: STRINGSINGLE, STRINGDOUBLE, COMMENTSINGLELINE, COMMENTMULTIPLELINE, vengono salvate le variabili yyline e yycolumn che verranno mostrate in caso di errore.

Nell’ultimo caso le due variabili responsabili dell’individuazione del carattere non riconosciuto sono immediatamente disponibili per l’utilizzo.

## ***Analizzatore Sintattico – CUP***

La grammatica fornita nelle specifiche del linguaggio Fun è stata modificata accertandosi di non modificare il linguaggio generato.   
È stato introdotto un nuovo non terminale “AssignStat” in modo da semplificare le produzioni: IdListInit, IdListInitObbl, Stat e aggiungere una nuova funzionalità al linguaggio, ovvero, la possibilità di creare una variabile di tipo VAR e assegnargli un’espressione.

Di seguito è sono riportati i non terminali modificati:

* IdListInit ::= ID I dListInit COMMA ID | AssignStat | IdListInit COMMA AssignStat
* IdListInitObbl ::= AssignStat | AssignStat COMMA IdListInitObbl
* Stat ::= IfStat SEMI | WhileStat SEMI | ReadStat SEMI | WriteStat SEMI | AssignStat SEMI   
  | CallFun SEMI | RETURN Expr SEMI
* AssignStat ::= ID ASSIGN Expr

È stata, poi, gestita l’eventualità in cui dopo il non terminale main viene inserito altro codice, eliminandolo e proseguendo alla compilazione del sorgente.

## ***Generazione dell’Abstract Syntax Tree***

Per la gestione degli alberi in questo progetto si è scelto di costruire la classe Node.java.

La particolarità di questa classe è la possibilità di generalizzazione, infatti, la stessa classe è stata utilizzata sia per la gestione dell’albero sintattico che per quello semantico.

La classe Node.java fa uso delle generics e contiene:

* Il riferimento al proprio genitore che nel caso della radice è uguale a null.
* Le informazioni relative al nodo, questa variabile è generic quindi può essere sostituita con qualsiasi tipo di variabile.
* Una lista di nodi che corrispondono ai figli del nodo che sarà vuota quando il nodo corrisponde ad una foglia dell’albero.
* Una serie di metodi per la gestione della classe stessa. Tra i metodi si può notare l’utilizzo dell’operatore “Three Dots” che permette di realizzare funzioni con un numero variabile di argomenti.

Dall’estensione di Node.java nasce la classe SintaticNode che eredita gli attributi e i metodi della classe genitore e fa uso variabile SintaticItem formata a sua volta da:

* Un nome, può assumere il valore di un terminale o un non terminale dell’analisi sintattica.
* Un possibile valore (nel caso di costanti o id).
* Il suo valore di ritorno.
* La Symbol Table relativa al nodo.

## ***Analizzatore Semantico***

L’analisi semantica viene effettuata utilizzando due visite dell’AST:

* La prima visita serve per:
  + Creare l’albero semantico, in cui ogni nodo gestisce un determinato scope del nostro programma e contiene una Symbol Table.
  + Associare una Symble Table ad ogni nodo dell’AST.
  + Controllare che le funzioni e le variabili vengano dichiarate prima dell’utilizzo.
  + Controllare che le funzioni effettuino il ritorno del parametro dove necessario.
* La seconda visita serve per:
  + Effettuare il controllo dei tipi
  + Inferire il tipo alle variabili sprovviste di un valore di ritorno.

## ***Generazione del codice intermedio***

Per ottenere un codice di maggiore comprensione e di facile gestione si è preferito creare una libreria Helper in C che viene compilata assieme al codice prodotto in questa fase.

Nello specifico questa libreria:

* Aggiunge un nuovo tipo “String” al linguaggio C permettendo di ridurre errori e controlli. L’aggiunta di questa libreria ha portato però ad un uso improprio della memoria che è stato risolto implementando un Garbage Collector che si occupa di gestire la memoria allocata.
* Gestisce le seguenti funzioni:
  + creaString()
  + confrontaString()
  + concat()
  + scan()
  + print()

Le seguenti funzioni sono state “sovraccaricate”, ogni metodo di questa libreria, è stato gestito con argomenti generic, così da utilizzare un unico nome di funzione con argomenti differenti. Per ottenere questo risultato si è utilizzato il pre processore del compilatore C e il tipo Generic.

# Regole di Type Checking

## ***Tipi Primitivi***

## ***Dichiarazioni di Variabili***

## ***Operazioni Unarie***

## ***Operazioni Binarie***

## ***Chiamata a Funzione senza ritorno***

## ***Chiamata a Funzione con ritorno***

## Statement

### ***If-Then***

### ***If-Then-else***

### ***While***

### ***Read***

### ***Read con Print***

### ***Print***

writeTerminal = {“?” , “?,” , “?:” , “?.”}

### ***Assign***

### ***Dichiarazione***

### ***Lista di Istruzioni***

## ***Tabelle delle operazioni***

|  |  |  |
| --- | --- | --- |
| OP | Operando 1 | Risultato |
| MINUS - PAR | Integer | Integer |
| MINUS - PAR | Real | Real |
| NOT | Boolean | Boolean |
| PAR | String | String |
| PAR | Boolean | Boolean |

|  |  |  |  |
| --- | --- | --- | --- |
| OP | Operando 1 | Operando 2 | Risultato |
| DIV | Real – Integer | Real – Integer | Real |
| PLUS-MINUS-TIMES-POW | Real – Integer | Real | Real |
| PLUS-MINUS-TIMES-POW | Real | Real – Integer | Real |
| PLUS-MINUS-TIMES-POW | Integer | Integer | Integer |
| DIVINT | Integer – Real | Integer – Real | Integer |
| STR\_CONCAT | String | String – Integer – Real | String |
| GT-GE-LT-LE | Integer – Real | Integer – Real | Boolean |
| EQ- NE | Integer | Integer | Integer |
| EQ- NE | Real | Real | Real |
| EQ- NE- AND-OR | Boolean | Boolean | Boolean |
| EQ- NE | String | String | String |