Università degli Studi di Salerno

Dipartimento di Informatica

Corso di Laurea Magistrale in Informatica

Software Engineering & IT Management

Linguaggio TOY2

Corso di Compilatori A.A. 2023/2024

Simona Grieco – 0522501126

Maria Concetta Schiavone – 0522501127

1) Introduzione

Tale documento è volto a fornire una descrizione completa di tutte le specifiche e regole del linguaggio “Toy2”, fornite durante il corso di Compilatori A.A 2023/2024.

2) Specifiche Lessicali

Tale sezione contiene la lista completa dei ***token*** rilevabili nel linguaggio Toy2 con le relative ***regole*** (pattern) utilizzate per il riconoscimento dei token corrispondenti.

Token	Pattern
VAR	var
TRUE	true
FALSE	false
REAL	real
INTEGER	integer
STRING	string
BOOLEAN	boolean
RETURN	return
FUNCTION	func
ENDFUNCTION	endfunc
PROCEDURE	proc
ENDPROCEDURE	endproc
OUT	out
DOLLARSIGN	$
IF	if
ELSE	else
THEN	then
ENDIF	endif
ELSEIF	elseif
WHILE	while
DO	do
ENDWHILE	endwhile
ENDVAR	endvar
REF	ref
BLOCK COMMENT	%
EQ	=
NE	<>
LT	<
LE	<=
GT	>
GE	>=
AND	&&

OR	||
NOT	!
PLUS	+
MINUS	-
TIMES	*
DIV	/
COLON	:
SEMI	;
COMMA	,
LPAR	(
RPAR	)
ASSIGN	^=
TYPERETURN	->
WRITE	-->
WRITERETURN	-->!
READ	<--
ID	[$_A-Za-z][$_A-Za-z0-9]*
STRING_CONST	\" ~\"
INTEGER_CONST	[0-9]+
REAL_CONST	[0-9]+(“.”[0-9]+)?

3) Specifiche Sintattiche

Tale sezione mostra:

- La ***grammatica*** senza conflitti utilizzata, dove i ***non terminali*** sono rappresentati in giallo e i ***terminali*** sono rappresentati in verde.
- La lista delle precedenze

Grammatica

```
Program ::= IterProc Procedure Iter

IterProc ::= VarDecl IterProc
          | Function IterProc
          | /* empty */

Iter ::= VarDecl Iter
      | Function Iter
      | Procedure Iter
      | /* empty */

VarDecl ::= VAR Decls

Decls ::= Ids COLON Type SEMI Decls
       | Ids ASSIGN Consts SEMI Decls
       | Ids COLON Type SEMI ENDVAR
```

```
      | Ids ASSIGN Consts SEMI ENDVAR

Ids ::= ID COMMA Ids
    | ID

Consts ::= Const COMMA Consts
       | Const

Const ::= REAL_CONST
      | INTEGER_CONST
      | STRING_CONST
      | TRUE
      | FALSE

Function  ::= FUNCTION ID LPAR FuncParams RPAR TYPERETURN Types COLON
Body ENDFUNCTION

FuncParams ::= ID COLON Type OtherFuncParams
           | /* empty */

OtherFuncParams ::= COMMA ID COLON Type OtherFuncParams
                | /* empty */

Type ::= REAL
     | INTEGER
     | STRING
     | BOOLEAN

Types ::= Type COMMA Types
      | Type

Procedure ::= PROCEDURE ID LPAR ProcParams RPAR COLON Body ENDPROCEDURE


ProcParams::= ProcParamId COLON Type OtherProcParams
          | /* empty */

OtherProcParams ::= COMMA ProcParamId COLON Type OtherProcParams
                | /* empty */

ProcParamId ::= ID
            | OUT ID

Body ::= VarDecl Body
     | Stat Body
     | /* empty */

Stat ::= Ids ASSIGN Exprs SEMI
     | ProcCall SEMI
     | RETURN Exprs SEMI
     | WRITE IOArgs SEMI
     | WRITERETURN IOArgs SEMI
     | READ IOArgs SEMI
     | IfStat SEMI
     | WhileStat SEMI
```

```
FunCall ::= ID LPAR Exprs RPAR
        | ID LPAR RPAR


ProcCall ::= ID LPAR ProcExprs RPAR
         | ID LPAR RPAR

IfStat ::= IF Expr THEN Body Elifs Else ENDIF


Elifs ::= Elif Elifs
      | /* empty */

Elif ::= ELIF Expr THEN Body


Else ::= ELSE Body
     |  /* empty */

WhileStat ::= WHILE Expr DO Body ENDWHILE

IOArgs ::= IOConc IOArgs
       |  DOLLARSIGN LPAR Expr RPAR IOArgs
       | /* empty */


IOConc ::= IOConc PLUS IOConc
       | STRING_CONST

ProcExprs::= Expr COMMA ProcExprs
         | REF ID COMMA ProcExprs
         | Expr
         | REF ID

Exprs ::= Expr COMMA Exprs
      | Expr

Expr ::=  FunCall
     | REAL_CONST
     | INTEGER_CONST
     | STRING_CONST
     | ID
     | TRUE
     | FALSE
     | Expr PLUS Expr
     | Expr MINUS Expr
     | Expr TIMES Expr
     | Expr DIV Expr
     | Expr AND Expr
     | Expr OR Expr
     | Expr GT Expr
     | Expr GE Expr
     | Expr LT Expr
     | Expr LE Expr
     | Expr EQ Expr
     | Expr NE Expr
     | MINUS Expr %prec UMINUS
```

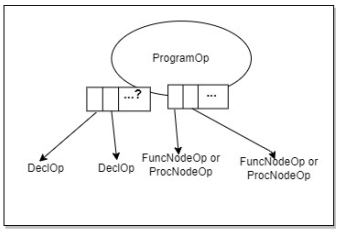
```
      | NOT Expr
      | LPAR Expr RPAR %prec ASSOC
```

Lista precedenze

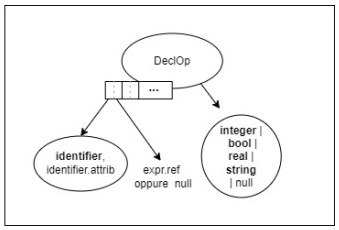
```
precedence left OR;
precedence left AND;
precedence right NOT;
precedence nonassoc EQ, NE, LT, LE, GT, GE;
precedence left PLUS, MINUS;
precedence left TIMES, DIV;
precedence left LPAR;
precedence left RPAR;
precedence left UMINUS;
precedence left ASSOC;
precedence left PROCEDURE;
```

4) Abstract Sintax Tree

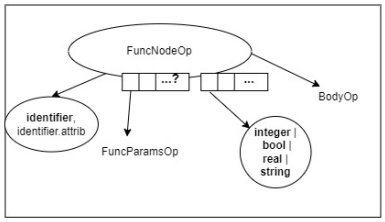
Nodo ProgramOp



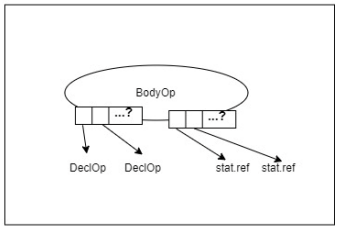
Nodo DeclOp



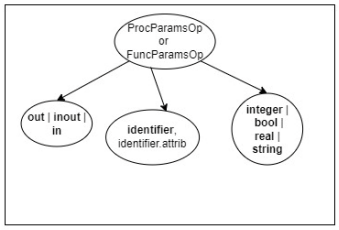
Nodo FuncNodeOp



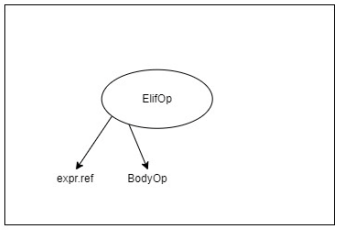
Nodo BodyOp



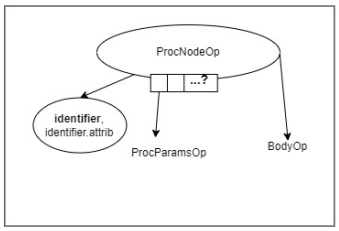
Nodo ProcParamsOp FuncParamsOp



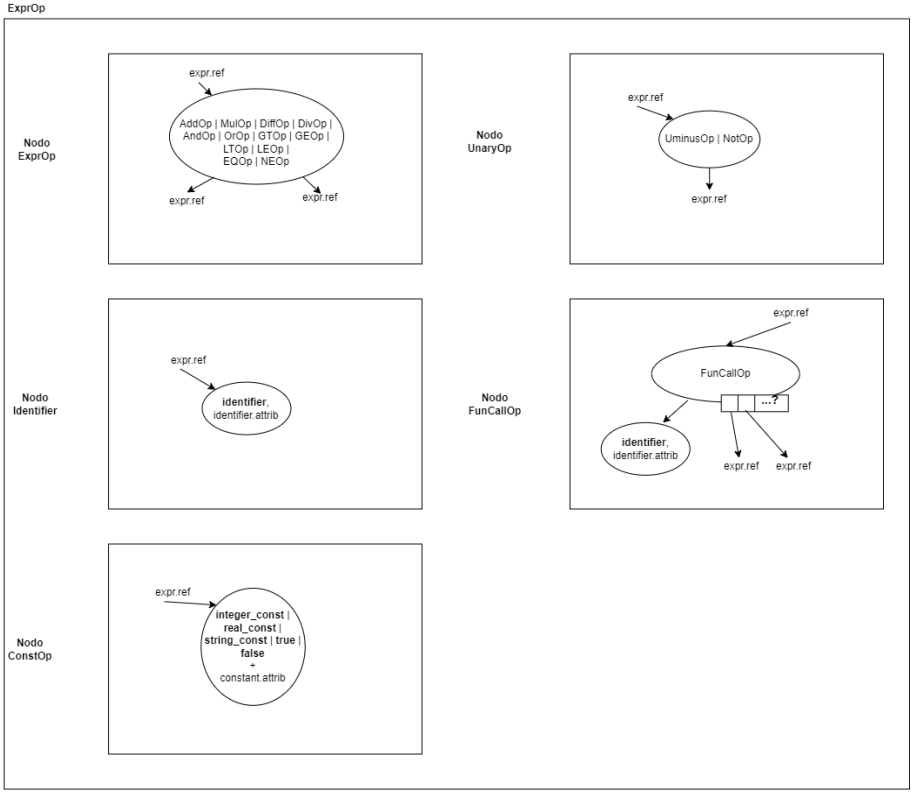
Nodo ElifOp



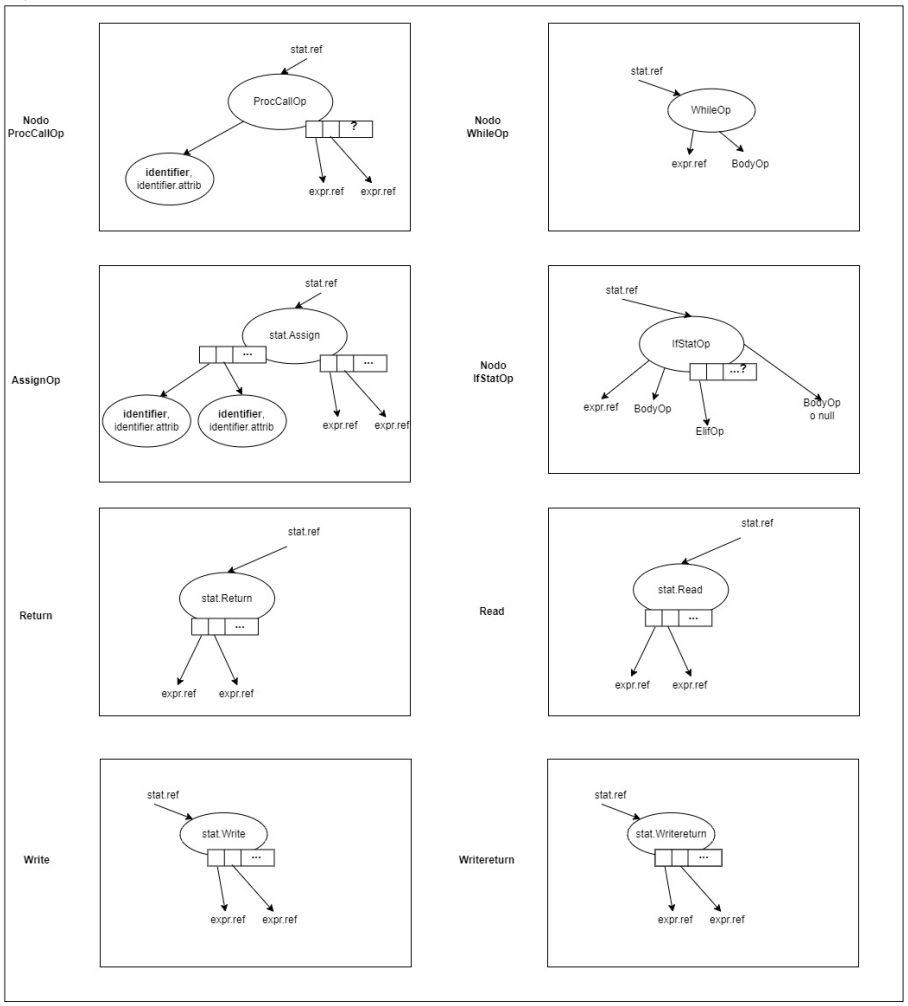
Nodo ProcNodeOp



Nodo ExprOp



Nodo StatOp



5) Specifiche Semantiche

Identificatore

$$\frac{\Gamma(id)=\tau}{\Gamma\vdash id:\tau}$$

Costanti

$$\Gamma \vdash \text{real\_const} : \text{real}$$
$$\Gamma \vdash \text{integer\_const} : \text{integer}$$
$$\Gamma \vdash \text{string\_const} : \text{string}$$
$$\Gamma \vdash \text{true} : \text{boolean}$$
$$\Gamma \vdash \text{false} : \text{boolean}$$

Lista di istruzioni

$$\frac{\Gamma\vdash stmt_1 : notype \; \Gamma\vdash stmt_2 : notype}{\Gamma\vdash stmt_1, stmt_2 : notype}$$

Chiamata a funzione

$$\frac{\Gamma\vdash f : \tau_1\times\ldots\times\tau_n -> \sigma_1\ldots\sigma_m \; \Gamma\vdash e_i : \tau_i^{i\in 1\ldots n}}{\Gamma\vdash f(e_1,\ldots,e_n) : \sigma_1\ldots\sigma_m}$$

Chiamata a procedura

$$\frac{\Gamma\vdash f : \tau_1\times\ldots\times\tau_n -> notype \; \Gamma\vdash e_i : \tau_i^{i\in 1\ldots n}}{\Gamma\vdash f(e_1,\ldots,e_n) : notype}$$

Assegnazione

$$\frac{\Gamma(id_i) : \tau_i^{i\in 1\ldots n} \; \Gamma\vdash e_i : \tau_i^{i\in 1\ldots n}}{\Gamma\vdash id_1,\ldots,id_n \wedge = e_1,\ldots,e_n : notype}$$

Dichiarazione-Istruzione

$$\frac{\Gamma[id\rightarrow\tau]\vdash stmt : notype}{\Gamma\vdash \tau\ id;\ stmt : notype}$$

While

$$\frac{\Gamma\vdash e : boolean \; \Gamma\vdash body : notype}{\Gamma\vdash \textbf{while}\ e\ \textbf{do}\ body\ \textbf{endwhile} : notype}$$

If, elseif, else

$$\frac{\Gamma \vdash e_1 : boolean\ \Gamma \vdash body_1 : notype\ \Gamma \vdash e_2, \ldots, e_n : boolean\ \Gamma \vdash body_2, \ldots, body_n : notype\ body_{n+1} : notype}{\Gamma \vdash \textbf{if}\ e_1\ \textbf{then}\ body_1\ \textbf{elseif}\ e_2\ \textbf{then}\ body_2\ \ldots\ \textbf{elseif}\ e_n\ \textbf{then}\ body_n\ \textbf{else}\ body_{n+1}\ \textbf{endif} : notype}$$

Write

$$\frac{\Gamma \vdash e : \tau}{\Gamma \vdash --> e : notype}$$

Read

$$\frac{\Gamma(id{=}\tau)\ \Gamma \vdash e : \tau}{\Gamma \vdash <-- id\ e : notype}$$

Return

$$\frac{\Gamma \vdash e_i : \tau_i^{i \in 1 \ldots n}}{\Gamma \vdash return\ e_1, \ldots, e_n : notype}$$

Operatori unari

$$\frac{\Gamma \vdash e : \tau_1 \quad optype1(op_1, \tau_1) = \tau}{\Gamma \vdash op1\ e : \tau}$$

Operatori binari

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_2 \quad optype2(op_2, \tau_1, \tau_2) = \tau}{\Gamma \vdash e_1\ op_2\ e_2 : \tau}$$

6) Test

Test	Output
invalid bad funcall	Il numero di parametri non coincide in Proc(main)!
invalid bad proc decl	La procedura non può contenere return in Proc(sommac)!
invalid bad return type	I tipi dei parametri in output non combaciano con quelli del return in Func(stampa)!
invalid bad write argument	Syntax error at character 18 of input instead expected token classes are [STRING_CONST, DOLLARSIGN, PLUS]
invalid fun redeclaration	Dichiarazione multipla!

invalid no out argument	Errore nei parametri per riferimento nella chiamata di procedura in Proc(main)!
invalid no out param	Alcuni parametri della chiamata a procedura sono stati passati erroneamente per riferimento in Proc(main)!
invalid no var declaration	Id non dichiarato: result
Invalid num param	Nella chiamata a procedura in While il numero di parametri non coincide
invalid out expression	Errore nei parametri per riferimento nella chiamata di procedura in While
invalid param type	I tipi dei parametri nella chiamata di funzione non coincide in Proc(main)
invalid return	Ci deve essere almeno un return nel corpo della funzione!
invalid return mult assign	Il numero di variabili usati per invocare la funzione non sono corretti in Else-If
invalid var mult assign	Id non dichiarato: sottrazione_res
invalid var mult assign2	Il numero degli id non è uguale al numero delle costanti in Func(tutte_le_operazioni)!
invalid var redeclaration	Dichiarazione multipla!
valid1	Pass
valid2	Pass
valid3	Pass
valid4	Pass
fibonacci	Pass
tabelline	Pass
test_1	Pass
test_2	Pass
test_3	Pass

7) Modifiche apportate

Sono state apportate alcune modifiche alla grammatica per risolvere dei conflitti:

Program ::= IterProc Procedure Iter

IterProc ::= VarDecl IterProc
| Function IterProc

```
Iter ::= VarDecl   Iter
       | Function  Iter
       | Procedure Iter


IOArgs ::= IOConc   IOArgs
         | DOLLARSIGN LPAR Expr RPAR IOArgs
         | /*empty */


IOConc ::= IOConc  PLUS  IOConc
          | STRING_CONST
```