

Modifiche del 6 dicembre ore 18:03

1. aggiunto descrizioni sul problema delle dichiarazioni var.
2. aggiunto sezioni (fine pagina) su chiamate a funzioni ed a procedure

Modifiche del 2 gennaio ore 10:40

1. è ora possibile avere usi anche di nomi locali dopo la loro dichiarazione.

DESCRIZIONE DELLE SPECIFICHE SINTATTICHE DEL LINGUAGGIO Toy2

NOTA BENE: Oltre le specifiche sintattiche questo documento riporta anche alcune specifiche semantiche racchiuse fra doppie quadre `*[[..]]*`. Si ricordi che le specifiche semantiche vanno IGNORED quando si scrive la grammatica.

IL PROGRAMMA

Il programma presenta dichiarazioni di variabili, funzioni e procedure in qualsiasi ordine.

Il programma deve avere almeno una procedura.

`*[[Una procedura deve chiamarsi "main"]]*`

LE DICHIARAZIONI DI VARIABILI

Le variabili possono essere dichiarate accorrandole sotto lo stesso tipo e separandole da virgole o, similmente, inizializzate a costanti.

`*[[Nell'inizializzazione il numero delle costanti deve essere pari al numero degli id]]*`

Le dichiarazioni iniziano con la parola chiave `var` e terminano con un backslash.

Esempi

```
var a, b: integer; y : real;\ %senza inizializzazione%
```

```
var x: real;\
```

Dichiarazioni di variabili possono presentare inizializzazione solo con una costante (in tal caso il tipo non deve essere presente)

```
var a ^= 3.2 ; c ^= 4; b, d ^= 1, 3; y, c: string;\ %sia  
dichiarazioni che inizializzazioni%
```

```
var x ^= "ciao";\
```

Sono errate le seguenti dichiarazioni:

```
var b;\ % manca il tipo %
```

```
var b; % manca la chiusura%
```

```
var a, b ^= 3;\ % l'assegnazione è scorretta semanticamente  
ma corretta sintatticamente - c'è una sola costante%
```

TYPES

I tipi utilizzabili sono: integer, real, string, boolean

FUNCTIONS

Le funzioni possono avere zero o più parametri in input ed UNO o più valori di ritorno; il suo corpo è preceduto dal carattere ":" e seguito da "endfunc". *[[Deve contenere l'istruzione return seguita dalle espressioni di ritorno (che devono essere di pari numero ai tipi di ritorno dichiarati).]]*

All'interno del corpo di una funzione o procedura le dichiarazioni di variabili possono essere alternate con le istruzioni. (L'uso di una variabile può non essere quindi preceduto da una sua dichiarazione *[[che deve comunque esserci]]*)

Il corpo può contenere una sola istruzione *[[che per le funzioni deve essere il return]]*

I parametri vanno dichiarati sempre singolarmente e separati da virgola.

Un esempio di dichiarazione di funzione è

```
func somma(a: integer, b: integer) -> integer :  
    var c ^= 0;\n    c = a+b;\n    return c;
```

```
endfunc
```

*[[

I parametri di una funzione sono solo in lettura e non possono essere mai assegnati (immutable).

La seguente dichiarazione di funzione è semanticamente errata anche se sintatticamente corretta

```
func somma(a: integer, b: integer) -> integer:
```

```
    a ^= a + b;      % a è di sola lettura e non può essere  
    assegnata %
```

```
    c ^= a + b;      % c è usata prima della sua  
dichiarazione%
```

```
    var c: integer;\n    return c;
```

```
endfunc
```

```
]]*
```

Una chiamata a funzione è sempre considerata un' espressione in quanto restituisce uno o più valori.

Esempi di espressioni sono: il numero 5, la moltiplicazione 3*2, la chiamata a funzione somma(4, 5) con valori, rispettivamente 5, 6 e 9.

Esempio corretto:

```
func coppia_1_2(boolean raddoppia) -> integer, integer:
```

```
    if raddoppia then  
        return 2, 4;
```

```
    else  
        return 1, 2;
```

```
endfunc
```

Altro esempio di funzione che restituisce valori di tipo diverso

```
func numero_positivo(a : integer)-> integer, boolean:
    var b : boolean;\
    if a > 0 then b ^= true else b ^= false endif;
    return a, b;
endfunc
```

Si noti che dopo "endfunc non c'è ";"

LE ASSEGNAZIONI

Un'assegnazione è espressa dal lessema ^= e può simultaneamente assegnare ad una o più variabili una o più espressioni (quando al di fuori di una dichiarazione).

*[[
il numero dei valori di ritorno delle espressioni deve essere uguale a quello delle variabili
]]*

Esempi corretti (anche semanticamente)

```
var a, b, c : integer;\
a, b ^= coppia_1_2(true); % assegnerà 2 in a e 4 in b %
var c : integer;\
var x : integer; y: boolean;\
x, y ^= numero_positivo(3);
a, b ^= 2*4, 5+2;
a, b, c ^= coppia_1_2(false), somma(coppia_1_2(true));
```

Esempi errati solo semanticamente

```
a ^= coppia_1_2(true);
a, b, c ^= 1, 2;
```

PROCEDURES

Sono come le funzioni ma non hanno valori di ritorno *[[ed i parametri di input sono sia in lettura che in scrittura (mutable) inoltre l'istruzione return non deve essere presente.]]* Inoltre i parametri di input possono essere qualificati come "out" che significa che il parametro è passato per riferimento.

Il corpo di una procedura può essere vuoto.

```
proc somma_proc(a: integer, b: integer): %versione non funzionante - manca out - ma corretta sintatticamente e semanticamente%
```

```
    a ^= a + b;      %corretto (siamo in procedura) - ma non produce output%
endproc
```

Le chiamate a procedura sono considerate come istruzioni alla stessa stregua di istruzioni quale l'assegnazione, l'if etc.

Infatti non prevedono tipo di ritorno.

Esempio di chiamata di una procedura

```

...
var a, b ^= 1, 2;\
somma_proc(a, b);
--> "La somma finale è" $(a);
% istruzione di stampa in output, equivalente a
printf("La somma finale è %d", a) %

```

L'output qui sarà 1 e non 3. Per stampare 3 la dichiarazione di somma dovrà usare l'out (passaggio per riferimento):

```

proc somma_proc(out a: integer, b: integer): % versione funzionante%
    a ^= a + b;
endproc

```

OUTPUT

L'output si esprime con l'operatore --> se non si vuole ritorno a capo e con l'operatore -->! per includere lo '\n' alla fine

L'espressione numerica da stampare va inclusa nel testo con l'operatore \$(..)

Esempio

```

--> "La somma finale è" $(a+b) "e va bene così";
% istruzione di stampa in output, equivalente a printf("La
somma finale è %d e va bene così", a+b) %
oppure (per il ritorno a capo)
-->! "La somma" + "finale è " $(a+b);
% istruzione di stampa in output, equivalente a printf("La
somma finale è %d\n", a+b) %

```

INPUT

L'input si esprime con l'operatore <--

L'id in cui leggere va incluso nel testo con l'operatore \$ (..)

Esempio

```

<- "Inserisci" + " un numero intero" $(a) "ed un numero reale"
$(b);
% è equivalente a printf("Inserisci un numero intero:\n");
scanf ("%d", &a); printf("ed un numero reale\n"); scanf ("%f",
&b);%
Si noti che lo "\n" è di default (informazione utile nella
fase di generazione del codice).

```

[[Le istruzioni di input e di output usano lo stesso operatore \$ ma si noti che, per l'output, l'argomento può essere una qualsiasi espressione, mentre per l'input, esso deve essere per forza un identificatore. Per semplificare la grammatica, questo controllo lo faremo nell'analisi semantica.]]

STATEMENTS

Queste includono la chiamata a procedura, l'assegnazione, i due modi di inviare output a video, la ricezione di input da tastiera, il return, il solito while e l'if con la variante elseif per casi a catena.

[[Si noti che le condizioni del while e dell'if potrebbero essere qualsiasi espressione ma la semantica vincolerà le espressioni ad essere booleane.]]

EXPRESSIONS

Queste includono la chiamata a funzione, tutte le costanti, gli identificatori *[[di variabili]]*, tutte le operazioni aritmetiche con operatori +, -, *, /, logiche con operatori &&, || e ! e di confronto =, <>, <, <=, >, >=.

Sintatticamente tutti gli operatori possono essere usati con qualsiasi ID o costante o funzione come argomento.

[[L'effettiva correttezza di uso di operatore ed argomenti è lasciato alla fase semantica]].

Ad esempio sono sintatticamente corrette (ma scorrette semanticamente) le espressioni:

```
coppia_1_2(true) + "ciao" > 2  
(a > 1) + 2
```

Sono sintatticamente e semanticamente corrette

```
(5 *4 +3) > 1  
((5 *4 +3) > a + b) && a > 1
```

Le precedenze ed associatività degli operatori aritmetici e logici sono quelle già note dalla matematica.

CHIAMATE A PROCEDURE

I parametri possono essere passati per riferimento, nel qual caso

1. il parametro deve essere un identificatore e non una espressione
2. l'identificatore parametro deve essere preceduta dal carattere @

CHIAMATE A FUNZIONI

I parametri non possono essere passati per riferimento