P O L I T E C N I C O D I B A R I

I Facoltà di Ingegneria

Dipartimento di Elettrotecnica ed Elettronica

Corso di Laurea Specialistica in Ingegneria Informatica

Tema d'Anno
in
Compilatori e Interpreti

Analizzatore Lessicale-Sintattico-Semantico MATLAB

Docente
Chiar.mo Prof.
Giacomo PISCITELLI

Studenti:
Domenico Maria AMORUSO
Silvia GIANNINI

Anno Accademico 2009-2010

INTRODUZIONE - OBIETTIVI **3**

1. PROGETTO DELL'INTERPRETE **3**

2. TOOL UTILIZZATI PER IL PROGETTO **5**

2.1 ANALISI LESSICALE 5
2.1.1 La struttura del programma generato *6*
2.2 ANALISI SINTATTICA 7
2.3 ANALISI SEMANTICA 9

3. GUIDA AL LINGUAGGIO **9**

3.1 IL MATLAB ARRAY 9
3.2 DATA STORAGE 9
3.3 TIPI DI DATI IN MATLAB 10
Matrici complesse in doppia precisione *10*
Matrici numeriche *10*
Stringhe MATLAB *10*
Array vuoti *10*
Variabili *11*
Numeri *11*
Operatori aritmetici *12*
Dichiarazione di matrici *13*

4. IMPLEMENTAZIONE **14**

5. COMANDI PER LA COMPILAZIONE E L'AVVIO **16**

APPENDICE **17**

SCANNER.L 17
PARSER.Y 19
SEMANTICA.H 43
Grammatica *55*
CASI DI TEST 57
def_variabili.txt *57*
op_aritmetiche_addizione.txt *58*
op_aritmetiche_sottrazione.txt *62*
op_aritmetiche_moltiplicazione.txt *66*
op_aritmetiche_divisione.txt *70*
trasposta e size.txt *73*
prova.txt *74*
errore.txt *76*

BIBLIOGRAFIA **78**

Introduzione - Obiettivi

Lo scopo del nostro progetto è stato quello di realizzare un interprete di comandi in linguaggio MATLAB, un linguaggio sviluppato per applicazioni specifiche.
Matlab è un software che utilizza un linguaggio di alto livello e che si è imposto come leader mondiale nel settore del calcolo ingegneristico e della simulazione. Il nome è la contrazione di **Mat**rix **Lab**oratory e questo dice già molto sull'impostazione generale del software, nel senso che la logica con cui opera è una logica prettamente matriciale. Una conoscenza almeno basilare di questo tipo di calcolo è quindi richiesta a chiunque abbia la necessità di lavorare con Matlab.
MATLAB è un ambiente software per il calcolo scientifico basato su un programma principale che, tramite un interprete, è in grado di eseguire istruzioni native (built-in) o contenute in files su disco (M-files).

Il software ha una struttura modulare nel senso che esistono una serie di applicazioni specifiche, denominate Toolboxes, che possono essere installate per ampliare le funzionalità di base. Ogni toolbox fornisce gli strumenti per la gestione di un problema specifico, come ad esempio: acquisizione e gestione di immagini, simulazione di sistemi, gestione di reti neurali e logica fuzzy, strumenti di ottimizzazione, matematica finanziaria e molto altro.

Matlab dispone anche, stabilmente integrato nel modulo principale, di un sistema molto sofisticato di gestione dei grafici sia in due che in tre dimensioni.

Abbiamo realizzato un traduttore che simula il comportamento del programma Matlab nella modalità di esecuzione file, in cui i comandi sono salvati all'interno di un M-file.

Gli M-files sono file di testo che contengono codice MATLAB. Possono essere creati utilizzando il MATLAB editor o un altro editor di testo per creare file che contengono gli stessi statements che si possono editare nella linea di comando MATLAB. Il file avrà estensione .m. Prenderemo in considerazione solo gli script MATLAB, che sono il più semplice tipo di M-file perchè non hanno né argomenti di input né di output. Essi sono utili per automatizzare una serie di comandi MATLAB che ad esempio devono essere ripetutamente eseguiti. Gli script condividono il workspace di base con la sessione MATLAB e gli altri script. Operano su dati esistenti nel workspace o possono creare nuovi dati su cui operare. Ciascuna variabile che lo script crea rimane nel workspace anche dopo il termine dello script in modo tale da poterla utilizzare in ulteriori computazioni. Nel momento in cui si esegue uno script, esso potrebbe sovrascrivere non intenzionalmente dati precedentemente memorizzati nel workspace di base.

1. Progetto dell'interprete

Un interprete è un programma che legge il programma sorgente e, per ogni istruzione, verifica la correttezza sintattica, effettua la traduzione nella corrispondente sequenza di istruzioni in linguaggio macchina ed esegue direttamente la sequenza di istruzioni. In tal modo è facile sviluppare i programmi ed eseguire il loro debug dal momento che la correttezza semantica viene effettuata per ogni istruzione; di contro istruzioni eseguite più volte (come ad esempio i cicli), vengono verificate e tradotte più volte e non è possibile riscontrare preventivamente la presenza di un errore sintattico finchè non si incontra l'istruzione errata. Di conseguenza, a fronte di un'alta facilità di messa a punto dei

programmi, la velocità di esecuzione di un interprete risulta bassa.

Un interprete, così come un compilatore, opera in due fasi:

-**Analisi**: crea una rappresentazione intermedia a partire dal programma sorgente attraverso l'analizzatore lessicale, sintattico ed infine semantico.

-**Sintesi**: crea il programma target equivalente a partire dalla precedente rappresentazione intermedia attraverso il generatore di codice intermedio, l'ottimizzatore ed il generatore di codice.

Durante queste fasi vengono utilizzati l'error handler e la symbol table. La tabella dei simboli contiene informazioni su tutti gli elementi simbolici incontrati quali: nome, *scope*, tipo etc. L'error handler comprende routine e procedure che gestiscono i vari errori.

Nel nostro progetto ci siamo occupati dello sviluppo delle fasi di analisi lessicale, sintattica e semantica dell'interprete.

Il lessico descrive le parole o elementi lessicali che compongono le frasi. L'**analisi lessicale**, effettuata dallo **scanner,** trasforma il codice sorgente a partire dall'individuazione dei token del linguaggio, ovvero gli elementi lessicali minimi non ulteriormente divisibili che descrivono un pattern, un insieme di caratteri con lo stesso significato (identificatori, costanti, delimitatori, operatori, caratteri compositi, keywords, numeri, etc). I tokens del linguaggio sono solitamente descritti attraverso espressioni regolari. Lo scanner, quindi, legge il programma sorgente carattere per carattere e ne ritorna i tokens, riconoscendo e segnalando eventuali errori lessicali; elimina eventuali informazioni non necessarie presenti nel codice sorgente (come ad esempio i commenti); processa le direttive di compilazione (include, define, etc).

L'**analisi sintattica**, svolta dal parser, è il procedimento di controllo sintattico e costruzione della derivazione di una frase rispetto ad una data grammatica, a partire dalla sequenza di token ricevuta in ingresso mediante lo scanner e da una specifica della sintassi. In generale quando lo scanner e il parser agiscono nello stesso tempo, un analizzatore lessicale non ritorna una lista di tokens, ma ritorna un token quando il parser lo richiede. Se le regole derivate dalla CFG sono soddisfatte, il risultato di questa fase è un **albero sintattico,** o **parse tree,** per il programma dato. In un parse tree, tutti i terminali sono foglie; tutti gli altri nodi sono non terminali.
Un analizzatore sintattico è, perciò, un algoritmo che opera su una stringa: se tale stringa appartiene al linguaggio associato alla grammatica, ne produce una derivazione (raggruppa i tokens in frasi grammaticali), altrimenti si ferma ed indica il punto della stringa dove si è presentato l'errore e il tipo di errore associato.

I parser si suddividono in due gruppi, in relazione a come viene creato il parse tree e al metodo di analisi sintattica:

Top-Down Parsing: *analisi sintattica discendente o "predittiva"*

Bottom-Up Parsing: *analisi sintattica ascendente o "a spostamento e riduzione"*

A questo scopo principale del parser, se ne aggiungono altri due non meno importanti:
- verificare se la sequenza di token rispetta le regole semantiche del linguaggio;
- costruire una rappresentazione intermedia (IR) del codice sorgente.

La sintassi, infatti, non basta a definire le frasi corrette di un linguaggio di programmazione, perché una stringa sintatticamente corretta non è detto che lo sia in assoluto, essa potrebbe violare altre condizioni non esprimibili nel modello noncontestuale. Da qui la necessità di attribuire un significato agli elementi sintattici e di stabilire delle regole semantiche alla composizione di elementi sintattici.

L'**analisi semantica** si occupa di controllare il significato delle istruzioni presenti nel codice in ingresso e di raccogliere informazioni necessarie per la generazione del codice. Controlli tipici di questa fase sono il type checking, ovvero controllare che gli identificatori siano stati dichiarati prima di essere usati, e cosi via. Il risultato di questa fase è l'albero sintattico astratto. Di norma le informazioni semantiche non possono essere rappresentate con un context-free language. Per fare questo le context-free grammars sono integrate con attributi (regole semantiche). Il risultato è una traduzione syntax-directed (traduzione diretta alla sintassi).

2. Tool utilizzati per il progetto

2.1 Analisi lessicale

Uno scanner può essere realizzato con differenti approcci. Noi ci siamo avvalsi di uno scanner generator, un programma in grado di generare automaticamente un analizzatore lessicale. Il prodotto FLEX (Fast LEXical analyzer generator), della Free Software Foundation, originariamente scritto in C da Vern Paxson nel 1987, è un free software, frequentemente usato con il parser generator Bison.

Flex legge il file di input (o lo standard input) con la descrizione dello scanner da generare. La descrizione in generale è fornita in forma di regole scritte come coppie di espressioni regolari e codice C. Questo file viene compilato e linkato assieme alla libreria -lfl per produrre un eseguibile. Flex è, quindi, un generatore che accetta in ingresso un insieme di espressioni regolari e di azioni associate a ciascuna espressione e produce in uscita un programma che riconosce tali espressioni.

Un file sorgente per FLEX è composto di tre sezioni distinte separate dal simbolo '**%%**':

definitions (#includes and scanner macros)
%%

Token definitions and actions
%%

C user code

Parti di codice possono essere inserite sia nella prima parte (inserendole tra i simboli **%{** e

%}) che nella seconda (tra **{ }** immediatamente dopo ogni espressione regolare che si vuole riconoscere), e vengono copiate integralmente nel file di output.

La prima sezione contiene le definizioni e può essere vuota. Ogni riga della prima sezione il cui primo carattere non sia di spaziatura è una definizione. Le definizioni consentono di dichiarare genericamente dei "name" secondo la forma

name definition

dove la definizione comincia al primo carattere diverso da blank che segue il nome e prosegue fino al termine della riga. Successivamente si potrà quindi far riferimento alla definizione mediante il suo nome. Le definizioni sono solitamente espresse mediante espressioni regolari.
La prima sezione può contenere, inoltre, le dichiarazioni in C per includere librerie e il file **parser.tab.h** prodotto da Bison, che contiene le definizioni dei token multi-caratteri, per definire variabili globali accessibili sia da yylex() che dalle routine dichiarate nella terza sezione e i prototipi delle funzioni dichiarate nella terza sezione.

La seconda sezione contiene le **regole** sotto forma di coppie

espressione_regolare (*pattern*) azione

che vengono riconosciute dall'analizzatore lessicale. Alcune utilizzano le definizioni della prima sezione, racchiudendone il nome tra parentesi graffe. Ciascun pattern ha un'azione corrispondente, espressa mediante qualsiasi statement in codice C, che viene eseguito ogni volta che un'espressione regolare viene riconosciuta. I pattern devono essere non indentati e le azioni devono iniziare sulla stessa riga in cui termina l'espressione regolare e ne sono separate tramite spazi o tabulazioni. Se tale codice comprende più di una istruzione o occupa più di una linea deve essere racchiuso tra parentesi graffe. Se l'azione è nulla (espressa con il carattere ';'), quando viene incontrato il token esso viene semplicemente scartato. Prima della prima regola si possono inserire istruzioni C (ad esempio commenti o dichiarazioni di variabili locali) ma solo se indentate o racchiuse tra %{}.

La terza sezione è opzionale e contiene le **procedure di supporto** di cui il programmatore intende servirsi per le azioni indicate nella seconda sezione: se è vuota, il separatore '%%' viene omesso. Tutte le righe presenti in questa sezione del programma sorgente sono ricopiate nel file **lex.yy.c** generato da Flex. Tale file è usato nelle routines che chiamano o sono chiamate dallo scanner.

(Per il file scanner.l si veda pag. 17 in APPENDICE)

2.1.1 La struttura del programma generato

Compilando il file FLEX con il comando **flex scanner.l** si ottiene come risultato il file lex.yy.c, un programma C, il cui **main()** contiene la routine di scanning **yylex()** assieme ad altre routine ausiliari e macro.

Per default:

```
int main()
      {
      yylex();
      return 0;
```

```
}
```

La funzione yylex() viene richiamata ripetutamente dalla funzione principale del parser, yyparse(). Quando l'eseguibile dello scanner è in esecuzione, la funzione yylex() ad ogni invocazione analizza l'input file yyin alla ricerca di occorrenze delle espressioni regolari e, nel momento in cui ne trova una, ritorna il prossimo token ed esegue il codice C corrispondente (l'azione), per poi proseguire nella scansione dell'input.
Se più di un pattern corrisponde all'input letto, Flex esegue l'azione associata all'espressione regolare che ha riconosciuto la sequenza più lunga o a quella dichiarata per prima in caso di pattern della stessa lunghezza, mentre se non viene trovata alcuna corrispondenza, si esegue la regola di default: il successivo carattere di input viene considerato matchato, ricopiando sul file yyout il testo non riconosciuto carattere per carattere. Per default, i file yyin e yyout sono inizializzati rispettivamente a stdin e stdout.
Se non specificato diversamente nelle azioni (tramite l'istruzione return), tale funzione (yylex()) termina solo quando l'intero file di ingresso è stato analizzato. Al termine di ogni azione l'automa si ricolloca sullo stato iniziale, pronto a riconoscere nuovi simboli.

Flex mantiene il testo, letto dall'input e riconosciuto da una espressione regolare, in un buffer accessibile all'utente tramite le variabili globali char *yytext e int yyleng, che contiene la sua lunghezza. Operando su tali variabili si possono definire azioni più complesse e passare al parser, attraverso la funzione **new_string,** il testo letto dall'input.

Al termine dell'input (EOF), FLEX (ed in particolare la funzione yylex()) invoca la function **yywrap()** a cui fornisce le variabili globali **char *yytext** e **int yyleng.** Se yywrap() ritorna 0, si assume che yyin stia puntando ad un nuovo input file e la scansione continua attraverso la yylex(); se ritorna 1, lo scanner termina e a sua volta restituisce 0 al programma che l'ha chiamato.

2.2 Analisi sintattica

È Bison a fornire le variabili per passare informazioni dallo scanner alla **yyparse()**. Ad ogni token, come pure ad ogni simbolo grammaticale, è possibile associare un valore. I valori associabili ad un token sono definiti da una **union**, dichiarata nel sorgente Bison. La variabile globale yylval, che è accessibile sia dallo scanner che dal parser, è utilizzata per passare a quest'ultimo il valore associato al token corrente.

Così come per lo scanner, esistono anche strumenti automatici per costruire automaticamente il compilatore di una grammatica. Esempi di generatori di questo tipo sono Yacc e Bison. Questi tool sono molto utili anche perché forniscono informazioni diagnostiche (ad esempio, in caso di grammatica ambigua, il tool riesce a determinare dove si crea l'ambiguità).
Il tool da noi utilizzato per generare il parser è Bison. Il programma Bison permette di descrivere le produzioni della grammatica del linguaggio da riconoscere e le azioni da intraprendere per ogni produzione. Il parser generato è di tipo bottom-up e la grammatica viene gestita con il metodo LR(1) (Left scan, Right-most derivation).
Un bottom-up parser cerca una right-most derivation del programma sorgente. Una "right-most derivation" è una derivazione nella quale durante ogni passo solo il non-terminale più a destra è sostituito. Il metodo di parsing Shift-reduce con un simbolo di lookahead (o LR(1)) genera una right most derivation:
• L indica che l'input viene letto da sinistra a destra (Left to right)
• R indica che viene ricostruita una derivazione Rightmost rovesciata (il non terminale più a destra viene sostituito per primo).

Bison genera una funzione per riconoscere l'input e utilizza l'analizzatore lessicale per prelevare dall'input i token e riorganizzarli in base alle produzioni della grammatica utilizzata. Quando una produzione viene riconosciuta, viene eseguito il codice ad essa associato.

Ogni programma Bison consta di tre sezioni:

Dichiarazioni

%%
Regole

%%
Sezione routines ausiliarie

La sezione delle regole è l'unica obbligatoria. I caratteri di spaziatura (blank, tab e newline) vengono ignorati. I commenti sono racchiusi, come in C, tra i simboli /* e */

Nella sezione dichiarazioni o definizioni si definiscono alcune informazioni globali utili per interpretare la grammatica.

Tramite l'istruzione:
%token NOMETOKEN
si definiscono quali sono i token inseriti nelle regole che sono il risultato dell'analisi lessicale.

Tramite l'istruzione:
%start assioma
si definisce qual è il non terminale della grammatica da considerare come assioma (per default il primo non terminale incontrato).

La sezione regole è composta da una o più produzioni espresse nella forma:
A : BODY ;
dove A rappresenta un simbolo non terminale e BODY rappresenta una sequenza di uno più simboli sia terminali che non terminali.
I simboli : e ; sono separatori. Nel caso la grammatica presenti più produzioni per lo stesso simbolo, queste possono essere scritte senza ripetere il non terminale usando il simbolo | .
Ad ogni regola può essere associata un'azione che verrà eseguita ogni volta che la regola viene riconosciuta. Le azioni sono istruzioni C e sono raggruppate in un blocco e possono apparire ovunque nel body di una regola.
Le azioni possono scambiare dei valori con il parser tramite delle pseudo-variabili introdotte dai simboli: $$, $1, $2, … La pseudo-variabile $$ è associata al lato sinistro della produzione mentre le pseudovariabili $n sono associate al non terminale di posizione n nella parte destra della produzione.

Bison si avvale della funzione yylex() dell'analizzatore lessicale per leggere l'input (ed eventuali valori) e convertirlo in token da passare al parser sotto forma di interi. Quindi parser ed analizzatore
lessicale devono accordarsi su quali valori rappresentano i token. L'accordo viene preso in modo automatico da Bison definendo i vari token tramite istruzioni C #define nel file **parser.tab.c** .

(Per il file parser.y si veda pag. 19 in APPENDICE)

2.3 Analisi semantica

Data la complessità della definizione della semantica di un linguaggio, la fase di analisi semantica dell'input non può essere automatizzata, come invece avviene per le fasi di analisi lessicale e analisi sintattica. Pertanto, il modulo che realizza la fase di analisi semantica è stato realizzato manualmente.
La gestione degli errori semantici rilevati durante il check sarà descritta più avanti nel paragrafo "Implementazione".

(Per il file semantica.h si veda pag. 43 in APPENDICE)

3. Guida al linguaggio

3.1 Il MATLAB Array

Il linguaggio MATLAB lavora con un unico tipo di oggetto: l'array MATLAB. In MATLAB tutte le variabili, inclusi scalari, vettori, matrici, stringhe, arrays di celle, strutture e oggetti vengono memorizzati come arrays MATLAB. In C, l'array MATLAB viene dichiarato di tipo mxArray. La struttura mxArray contiene, principalmente:

- il tipo associato all'array
- la dimensione in termini di numero di righe e colonne
- i dati associati all'array
- se numerico, il tipo della variabile (reale o complessa)
- se una matrice sparsa, gli indici degli elementi diversi da zero
- se una struttura o un oggetto, il numero dei campi e il nome per ciascun campo.

Nel nostro tema d'anno, allora, abbiamo associato all'array MATLAB una struttura in C, semplificata rispetto alla versione proposta da MATLAB. Tale struttura, indicata con il nome symrec per analogia con un record della tabella dei simboli, contiene:

- il nome del simbolo di variabile
- il tipo associato all'array
- la dimensione in termini di numero di righe e colonne
- la lista di valori associati all'array indicato dal simbolo

3.2 Data Storage

Tutti i dati in MATLAB vengono memorizzati per colonne, secondo una convenzione di memorizzazione delle matrici ereditata dal linguaggio Fortran. Questa osservazione è molto importante soprattutto in riferimento alle matrici di stringhe. Per esempio, data la

matrice:

```
a=['house'; 'floor'; 'porch']
a =
   house
   floor
   porch
```

le sue dimensioni sono:

```
size(a)
ans =
     3     5
```

3.3 Tipi di dati in MATLAB

Matrici complesse in doppia precisione

Il tipo di dato più comune in MATLAB è il tipo complesso in doppia precisione. I MATLAB arrays risultano quindi di tipo double, di dimensione m-per-n righe/colonne. I dati vengono memorizzati come due vettori in doppia precisione, uno contenente la parte reale dei dati e l'altro quella immaginaria, a cui si fa riferimento mediante i due puntatori pr e pi rispettivamente. Un array MATLAB reale verrà quindi rappresentato come una matrice in doppia precisione in cui il puntatore pi è NULL.

Matrici numeriche

Il MATLAB supporta anche altri tipi di dati numerici, da noi utilizzati. Questi sono: reali in singola precisione, interi a 16-bit, sia signed che unsigned. Nel nostro caso ci siamo soffermati solo su matrici reali, di conseguenza i dati verranno memorizzati in un singolo vettore, che nel nostro particolare caso è stato implementato come una lista i cui elementi sono di tipo *char, in modo tale da rendere indipendente dal tipo di dato la memorizzazione dei singoli elementi di una matrice all'interno della tabella dei simboli.

Stringhe MATLAB

In MATLAB le stringhe vengono individuate dal tipo di dato char e memorizzate come interi a 16 bit unsigned, senza parte immaginaria. Nel nostro caso, come precedentemente introdotto, tutti gli elementi verranno memorizzati come stringhe di caratteri (preservando sempre nel campo tipo il tipo di dato associato) e nel caso particolare delle stringhe verrà memorizzata la dimensione dell'array secondo la convenzione già introdotta.

Array vuoti

Gli array MATLAB di qualsiasi tipo possono essere vuoti. Un mxArray vuoto è una struttura con almeno una dimensione pari a 0. Per esempio un mxArray in doppia precisione con m ed n uguali a 0 prevede puntatore a NULL.

Abbiamo quindi adottato la convenzione di memorizzare gli array vuoti come array di tipo null con dimensioni in termini di righe e colonne pari a 0 e puntatore alla lista di valori a NULL.

Variabili

Come molti linguaggi di programmazione, il MATLAB supporta espressioni matematiche ma, differentemente da molti linguaggi di programmazione, queste espressioni coinvolgono intere matrici.

Il MATLAB non richiede alcun tipo di istruzione di dichiarazione o di dimensionamento delle variabili. Nel momento in cui il MATLAB incontra un nuovo nome di variabile, automaticamente crea la variabile e alloca l'appropriata dimensione di memoria. Se la variabile è già esistente, MATLAB cambia il suo contenuto e se necessario alloca nuova memoria. Per esempio:

```
num_students = 25
```

crea un matrice 1-per-1 chiamata num_students e memorizza il valore 25 nel suo singolo elemento.

Nell'implementazione del parser, nel momento in cui nell'M-file compare l'istruzione di assegnazione di una variabile, viene richiamata la funzione get_sym del file semantica.h che verifica l'eventuale preesistenza del simbolo all'interno della symbol table. Se il simbolo è inesistente viene semplicemente aggiunto assieme ai valori degli altri campi all'interno della tabella dei simboli, altrimenti viene stampato a video un messaggio che avvisa l'utente che il valore del simbolo è stato ridefinito e il campo della symbol table relativo al simbolo viene interamente sovrascritto, perdendo l'informazione precedente.

Per visualizzare a video il valore di ciascuna variabile (matrice, vettore o scalare) bisogna semplicemente immettere il nome della variabile.

I nomi di variabile devono sempre iniziare con una lettera, che può essere seguita da un qualsiasi numero di lettere, numeri o underscores. Per questo nello scanner ritroviamo la definizione del token

```
identificatore          [A-Za-z][A-Za-z0-9_]*
```

Il MATLAB è case sensitive per cui distingue tra lettere maiuscole e minuscole: A e a non sono la stessa variabile.

Sebbene i nomi di variabile possano essere di qualsiasi lunghezza, il MATLAB utilizza solo i primi N caratteri, dove N è il numero restituito dalla funzione namelengthmax (solitamente 63), e ignora i restanti. Per questo è importante rendere ogni nome di variabile unico rispetto agli altri nei primi N caratteri. Nel nostro interprete questo aspetto è stato trascurato.

Un tipo particolare di variabile è la variabile ans: most recent ANSwer. Il software MATLAB crea la variabile ans automaticamente quando non si specifica l'argomento di output.

Numeri

MATLAB utilizza per i numeri la notazione decimale convenzionale, con il punto prima delle cifre decimali. All'inizio può essere indicato il segno + o il segno - .

Alcuni esempi di numeri legali sono:

```
3               -99             0.0001    9.6397238
```

Nello scanner abbiamo inserito le definizioni dei tokens:

```
unsigned_number     [0-9]+
floating            (([0-9]+\.[0-9]*)|([0-9]*\.[0-9]+))
```

Operatori aritmetici

Gli operatori aritmetici si prestano alla computazione numerica. Le espressioni da noi formalizzate utilizzano i comuni operatori aritmetici e le note regole di precedenza.

+	Addizione
-	Sottrazione
*	Moltiplicazione
/	Divisione
‘	Trasposta complessa coniugata

Fatta eccezione per alcuni operatori matriciali, gli operatori aritmetici in MATLAB lavorano sugli elementi corrispondenti degli array con uguali dimensioni. Per vettori e array rettangolari, entrambi gli operandi devono essere della stessa dimensione a meno che uno dei due non sia scalare. Se un operando è scalare e l’altro no, MATLAB applica lo scalare a ciascun elemento dell’altro operando secondo una proprietà nota come espansione scalare.
L’esempio utilizza l’espansione scalare per elaborare il prodotto di uno scalare con una matrice:

```
A =
     8     1     6
     3     5     7
     4     9     2

3 * A
ans =
    24     3    18
     9    15    21
    12    27     6
```

Se le dimensioni sono incompatibili, il MATLAB restituisce un errore:

```
Error using ==> +
Matrix dimensions must agree.
```

Nelle matrici reali, l’operazione di trasposizione scambia gli elementi a_{ij} e a_{ji}. Dal momento che opereremo solo su matrici reali l’operazione di trasposizione complessa coniugata coincide con la trasposizione senza coniugazione.

- **Precedenza degli operatori**

Si possono costruire espressioni aritmetiche che utilizzano una qualsiasi combinazione degli operatori aritmetici precedentemente introdotti. I livelli di precedenza determinano l’ordine in cui MATLAB valuta un’espressione. All’interno di ciascun livello di

precedenza, gli operatori hanno uguale precedenza e vengono valutati da sinistra a destra. Le regole di precedenza degli operatori MATLAB sono mostrate in questa lista, ordinate dal più alto livello di precedenza a più basso:

1. trasposta complessa coniugata(')
2. moltiplicazione matriciale (*) , divisione matriciale (/)
3. addizione(+) , sottrazione(-)

- **Operatori aritmetici matriciali**

+ `A+B` somma A e B. A e B devono avere le stesse dimensioni, a meno che un addendo non sia uno scalare; in tal caso esso può essere sommato ad una matrice di qualunque dimensione.

- `A-B` sottrae B da A. A e B devono avere le stesse dimensioni, a meno che B non sia uno scalare; in tal caso esso può essere sottratto da una matrice di qualunque dimensione.

* `C = A*B` moltiplica le matrici A e B. Il numero di colonne di A deve essere uguale al numero di righe di B. Uno scalare può moltiplicare una matrice di qualsiasi dimensione.

/ `B/A` con A scalare restituisce una matrice i cui elementi sono gli elementi di B divisi per lo scalare A.

' `A'` effettua la trasposizione di matrice scambiando le righe con le colonne.

Dichiarazione di matrici

In MatLab una matrice viene dichiarata indicandone gli elementi tra parentesi quadre, separando gli elementi di una riga con blank o virgole e utilizzando puntoevirgola (;) per indicare la fine di ciascuna riga, come ad esempio:

```
A = [16 3 2 13; 5 10 11 8; 9 6 7 12; 4 15 14 1]
```

Un vettore colonna è una matrice *m*-per-1, un vettore riga è una matrice 1-per-*n*, uno scalare è una matrice 1-per-1. Le istruzioni

```
u = [3; 1; 4]

v = [2 0 -1]

s = 7
```

producono:

```
u =
        3
        1
        4
```

```
v =
        2       0      -1

s =
        7
```

4. Implementazione

Gli analizzatori da noi realizzati sono in gradi di riconoscere ed eseguire un subset ridotto di istruzioni in linguaggio MatLab ed in particolare:

1. Definizione di variabili, vettori e matrici di tipo numerico e alfanumerico

Es: Definizione di varabile:
a = 0.1 ;
d = a;
Definizione di vettore:
b = [1 2 3];
Definizione di matrice:
c = [1 2 3 ; 4 5 6] ;

Non è prevista la definizione di vettori i cui elementi sono nomi di variabili e la definizione di variabili ans con vettori. In tal caso verrà sollevato errore sintattico.
L'analisi semantica viene utilizzata per questi tipi di statement per verificare la presenza nella symbol table di variabili con lo stesso identificatore.

2. Stampa a video dei valori delle variabili scalari, dei vettori e delle matrici utilizzando semplicemente il nome identificativo della variabile.

Nel caso venga richiesta una variabile non dichiarata il programma mostra un messaggio di errore

Es: e
Error: Undefined function or variable

3. Operazioni di prodotto tra vettore (variabile) e matrice (variabile), matrice (variabile) e matrice (variabile), tra scalare (variabile o numero) e matrice (variabile), tra scalare (variabile o numero) e vettore (variabile), tra scalare e scalare (entrambi variabili o numeri).

Qualora una variabile sia associata ad una stringa di caratteri l'analizzatore semantico dell'interprete fornirà un messaggio d'errore.
Il risultato di un prodotto deve essere necessariamente assegnato ad una variabile.
Per fare il prodotto tra 2 matrici (vettori) si è usato il prodotto riga-colonna. Tale operazione fornirà un errore qualora il numero di righe della seconda matrice(vettore) non coincida col numero di colonne della prima matrice(vettore).

Es: a = [1 ; 2]
a =
1
2

b = [1 2 3]
b =
1 2 3

```
d = a*b
d =
        1 2 3
        2 4 6

c = b*a
Error using mtimes: Inner matrix dimensions must agree
```

4. Operazioni di somma e sottrazione tra variabili a cui sono associate matrici, vettori o scalari e variabili, tra variabili a cui sono associate matrici, vettori o scalari e numeri e tra numeri e numeri. Qualora una variabile sia associata ad una stringa di caratteri l'analizzatore semantico dell'interprete fornirà un messaggio d'errore.
Il risultato di una somma o sottrazione deve essere necessariamente assegnato ad una variabile.
Il compilatore fornirà un messaggio d'errore qualora le dimensioni delle matrici coinvolte nell'operazione non siano corrette.

Es:
```
c = b+b;
c =
        2 4 6

d = b+2 ;
d =
        3 4 5
```

5. Operazioni di divisione ma solo tra matrice(variabile) e scalare(variabile o numero), tra vettore(variabile) e scalare(variabile o numero) e scalare(variabile o numero) e scalare(variabile o numero).
Qualora una variabile sia associata ad una stringa di caratteri l'analizzatore semantico dell'interprete fornirà un messaggio d'errore.
Il risultato di una divisione deve essere necessariamente assegnato ad una variabile.
Il compilatore fornirà un messaggio d'errore quando si compie una divisione per 0 o per una variabile a cui è associato un vettore o una matrice.

Es:
```
e=b/2
e =
0.5000 1.0000 1.5000
```

6. Operazione di trasposizione

Es:
```
a = [1 2;3 4;5 6]
a =
        1 2
        3 4
        5 6

f = a'
f =
        1 3 5
        2 4 6
```

7. Comando size: indica le dimensioni di una matrice, di un vettore e di una variabile in

termini di numero di righe e numero di colonne.
Es: a = [1 2 3]
a =
[1 2 3]
size(a)
ans =
1 3

8. Comando clear: elimina tutte le variabili memorizzate fino a questo momento.

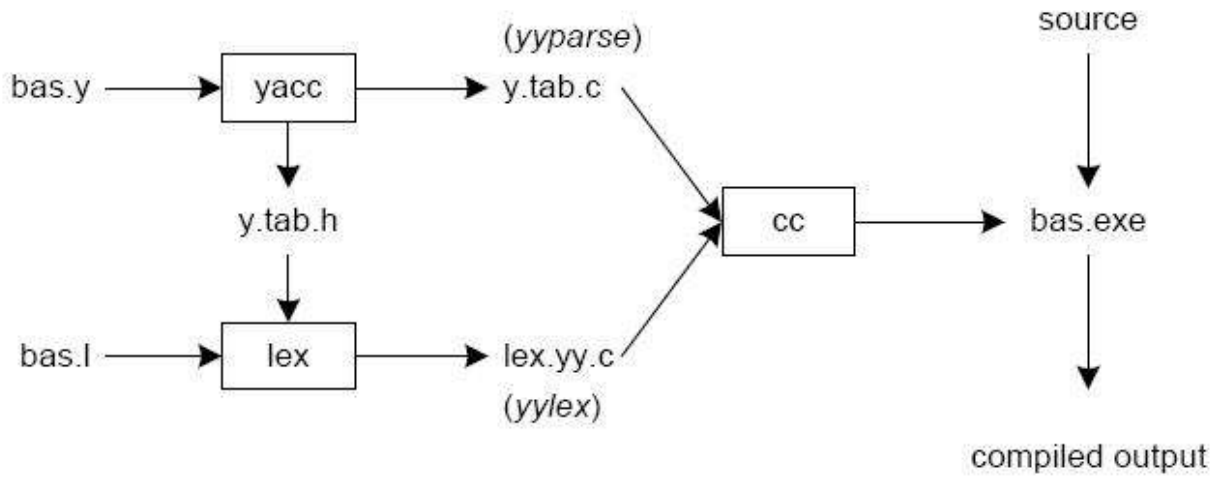
Il carattere di terminazione di un'operazione è il carattere di new-line (\n | \r | \r\n) preceduto o meno dal simbolo ";" .

In APPENDICE sono riportati i vari casi di test, sia che terminino con successo sia che segnalino errore.

5. Comandi per la compilazione e l'avvio

Di seguito sono riportati i comandi necessari per la compilazione e l'avvio, dal prompt dei comandi, dell'analizzatore da noi realizzato. Si suppone che siano stati preventivamente installati sulla macchina i software Flex, Bison e gcc e che nella variabile di sistema "Path" sia stata inserita la seguente riga: ;C:\Programmi\GnuWin32\bin; C:\MinGw\bin;
In maniera tale da poter compilare i sorgenti semplicemente selezionando il path contenente questi ultimi.

bison –d parser.y	crea parser.tab.h e y.tab.c
gcc -c parser.tab.c	crea parser.tab.o
flex scanner.l	crea lex.yy.c
gcc -c lex.yy.c	crea lex.yy.o
gcc parser.tab.o lex.yy.o –o compiler.exe	compila e linka
compiler.exe nomefile.txt	analizza il file nomefile.txt



APPENDICE

scanner.l

```
/*Definizioni*/

%{
      #include "parser.tab.h"
      #include <stdio.h>
      #include <stdlib.h>
      #include <string.h>
      char vett[255];
      void commento();
%}

/* %option main */
%option noyywrap

identificatore    [A-Za-z][A-Za-z0-9_]*
unsigned_number   [0-9]+
floating    (([0-9]+\.[0-9]*)|([0-9]*\.[0-9]+))
str_char    \'[\ A-Za-z0-
9\|!"£\$\%&\/\(\)=\?\^\\ì€éè\[\*\+\]çò@°à#$ù\<\>;:_,\.\-\{\}]*\'

%%

      /*Definizioni dei tokens e azioni*/

"%"         { commento(); }

" "         ;
"\r\n"            ;
"\r"        ;
"\n"        return NEWLINE;

"="             return UGUALE;
";"             return PUNTOEVIRGOLA;
"["         return QUADRAAPERTA;
"]"         return QUADRACHIUSA;
"("         return TONDAAPERTA;
")"         return TONDACHIUSA;
","         return VIRGOLA;

"'"             return APICE;
"*"         return PER;
"+"         return PIU;
"-"         return MENO;
"/"         return DIVISO;

"size"            return SIZE;
"clear"           return CLEAR;


{identificatore}  {
                        yylval.id = (char *)strdup(yytext);
                        return ID;
                  }

{unsigned_number} {
                        yylval.intval=atoi(yytext);
                        return UNSIGNED_NUMBER;
```

```
                  }

{floating}        {
                        strcpy(vett,&yytext[0]);
                        yylval.float_value=atof(yytext);
                        return FLOAT_NUMBER;
                  }

{str_char}        {
                        strcpy(vett,&yytext[0]);
                        yylval.id = (char *)strdup(vett);
                        return STR_CHAR;
                  }

%%

/*Codice C utente*/

void commento()
{
      char c;
      putchar('%');

      while ((c=input()) != '\n' && c != 0)
            putchar(c);

      if (c!=0)
            printf("\n");
}
```

parser.y

```
%{
      #include <stdio.h> /*Serve per le operazioni di I/O*/
      #include <stdlib.h> /* Serve per allocare memoria nella tabella dei
simboli*/
      #include <string.h> /* Serve per la strcmp  nella tabella dei
simboli */
      #include <ctype.h>
      #include "semantica.h" /* Tabella dei simboli */

      int yylex(void);
      void yyerror(char *);

      char *tmp=NULL, *var=NULL;
      int tipo=0, type=0, typeA;
      char *string=NULL;

      int i;
      int d;
      int riga;
      int n=0;
      int l;
      int n_righe=1;
      int n_colonne=1;
      int vrighe=1;
      int vcolonne=1;
      int vrighe1;
      int vcolonne1;
      int vrigheA;
      int vcolonneA;
      int fr=1;
      int opi;
      float opf;
      int op1i;
      float op1f;
      int op2i;
      float op2f;
      int opAi;
      float opAf;
      int opBi;
      float opBf;

      char end[1] = "/";

      symrec *s;

      struct elementoLista *lista = NULL; // puntatore della lista

      struct elementoLista *h = NULL; // puntatore della lista ausiliaria
      struct elementoLista *ris=NULL; // puntatore della lista ausiliaria

      struct elementoLista *op=NULL;
      struct elementoLista *op1=NULL;
      struct elementoLista *op2=NULL;
      struct elementoLista *opA=NULL;
      struct elementoLista *opB=NULL;

      int controllo_type(int type,int tipo,char *tmp,char *var,int l,int
n,int riga);

%}
```

```
%union semrec {
      int intval; /* Integer values */
      char *id; /* Identifiers */
      float float_value;
}

%start input

%token UGUALE APICE PUNTOEVIRGOLA VIRGOLA
%token QUADRAAPERTA QUADRACHIUSA TONDAAPERTA TONDACHIUSA
%token NEWLINE
%token PER PIU MENO DIVISO
%token <intval> UNSIGNED_NUMBER
%token <float_value> FLOAT_NUMBER
%token <id> ID STR_CHAR
%token SIZE CLEAR
%left PIU MENO
%left PER DIVISO

%%

/* Simbolo iniziale della grammatica */
input: /**/ |
      input start{
            printf("\n");
      }
;

start: statement_list PUNTOEVIRGOLA NEWLINE{
      printf(";\n");
      lista=NULL;
      type=0;
}
|statement_list NEWLINE{
      printf("\n");
      lista=NULL;
      type=0;
}
|NEWLINE {
      printf("\n");
      lista=NULL;
      type=0;
}
;

statement_list: statement_list PUNTOEVIRGOLA{
      printf("; ");
      lista=NULL;
      type=0;
} stm
|statement_list VIRGOLA{
      printf(", ");
      lista=NULL;
      type=0;
} stm
|stm {lista=NULL;type=0;}
/*Gestione errori*/
|stm " " stm
{
      yyerror("\nError: Unexpected MATLAB expression.\n");
      exit(0);
}
/*Fine gestione errori*/
```

```
;

stm: ID UGUALE {
      printf("%s",$1);
      printf(" = ");
} assignment{

      if(tipo==0)
            install($1,null,lista,0,0);
      if (tipo==1)
            install($1,INT,lista,n_righe,n_colonne);
      if (tipo==3)
            install($1,FLOAT,lista,n_righe,n_colonne);
      if (tipo==4)
            install($1,CHAR,lista,n_righe,n_colonne);
      n_righe=1;
      n_colonne=1;
      vrighe=1;
      vcolonne=1;
      type=0;
      tipo=0;
      lista=NULL;
      tmp=NULL;
      var=NULL;
      ris=NULL;
      n=0;
      fr=1;
}

|vet_list {
      install("ans",tipo,lista,n_righe,n_colonne);
      lista=NULL;
      type=0;
}

|ID{
      printf("%s",$1);
      s=NULL;
      s=(symrec *)malloc(sizeof(symrec));
      s=getsym ($1);
      if (s==0){
            yyerror("\nError: Undefined function or variable.\n");
            exit(0);
      }
}

|SIZE TONDAAPERTA ID TONDACHIUSA{
      printf("\nsize");
      printf("(");
      printf("%s",$3);
      printf(")");
      s=NULL;
      s=(symrec *)malloc(sizeof(symrec));
      s=getsym ($3);
      if (s==0){
            yyerror("\nError: Undefined function or variable.\n");
            exit(0);
      }
      else{
            n_righe = s->n_righe;
            n_colonne = s->n_colonne;
            printf("\nans =\n%d\t%d",n_righe,n_colonne);
      }
}
```

```
|SIZE TONDAAPERTA {printf("\nsize"); printf("(");} vet_list TONDACHIUSA{
      printf(")");
      printf("\nans =\n%d\t%d",n_righe,n_colonne);
}

|CLEAR {
      printf("clear\n");
      clear();
}

/*Gestione errori sintattici*/
| vet_list {printf(" ");} vet_list{
      yyerror("\nError: Unexpected MATLAB expression.\n");
      exit(0);
}
| vet_list ID{
      printf(" %s",$2);
      yyerror("\nError: Unexpected MATLAB expression.\n");
      exit(0);
}
| ID {printf("%s ",$1);} vet_list{
      yyerror("\nError: Undefined function or method 'k' for input
arguments of type 'char'.\n");
      exit(0);
}
| ID ID{
      printf("%s ",$1);
      printf("%s",$2);
      yyerror("\nError: Undefined function or method 'k' for input
arguments of type 'char'.\n");
      exit(0);
}
| vet_list errore {
      yyerror("\nError: The expression to the left of the equals sign is
not a valid target for an assignment.\n");
      exit(0);
}
| ID UGUALE{
      printf("%s =",$1);
      yyerror("\nError: Expression or statement is incomplete or
incorrect.\n");
      exit(0);
}
|ID UGUALE APICE {
      printf("%s = '",$1);
      yyerror("\nError: A MATLAB string constant is not terminated
properly.\n");
      exit(0);
}
| errore {
      yyerror("\nError: The expression to the left of the equals sign is
not a valid target for an assignment.\n");
      exit(0);
}
|QUADRAAPERTA {printf("[");}row_lists QUADRACHIUSA{
      printf("]");
      yyerror("\nError: The expression is not allowed.\n");
      exit(0);
}
/*Fine gestione*/

;

assignment:QUADRAAPERTA{
```

```
      printf("[");
      riga = 0;
      i = 0;
      type=0;
      fr=0;
} row_lists QUADRACHIUSA{
      printf("]");
      n_righe=riga;
}

|QUADRAAPERTA QUADRACHIUSA{
      printf("[");
      printf("]");
      riga = 0;
      i = 0;
      tipo=0;
      lista=aggiungi_elemento(lista,"NULL");
}

| op_arit

;

errore: UGUALE {printf("=");}
      | UGUALE {printf("=");} vet_list
      | UGUALE ID {printf("="); printf("%s",$2);}

;

row_lists :row_lists PUNTOEVIRGOLA {printf(";
");n=0;i=0;n_colonne=0;tmp=NULL;} vet_lists{
      riga++ ;
      controllo_dim(i,riga);
      n_colonne=i;
      i=0;
      if(tipo==4){
            int q=strlen(string);
            int ind;
            for(ind=0;ind<q;ind++){
                  char *stringT=NULL;
                  stringT=malloc(sizeof(char));
                  strncpy(stringT,&string[ind],1);
                  lista=aggiungi_elemento(lista,stringT);
                  stringT=NULL;
            }
      }
      lista = aggiungi_elemento(lista,(char*)end);
      n=0;
      tmp=NULL;
}

|row_lists NEWLINE {printf("\n");n=0;i=0;n_colonne=0;tmp=NULL;}
vet_lists{
      riga++ ;
      controllo_dim(i,riga);
      n_colonne=i;
      i=0;
      if(tipo==4){
            int q=strlen(string);
            int ind;
            for(ind=0;ind<q;ind++){
                  char *stringT=NULL;
                  stringT=malloc(sizeof(char));
                  strncpy(stringT,&string[ind],1);
```

```
                  lista=aggiungi_elemento(lista,stringT);
                  stringT=NULL;
            }
      }
      lista = aggiungi_elemento(lista,end);
      n=0;
      tmp=NULL;
}

|row_lists PUNTOEVIRGOLA NEWLINE
{printf(";\n");n=0;i=0;n_colonne=0;tmp=NULL;} vet_lists{
      riga++ ;
      controllo_dim(i,riga);
      n_colonne=i;
      i=0;
      if(tipo==4){
            int q=strlen(string);
            int ind;
            for(ind=0;ind<q;ind++){
                  char *stringT=NULL;
                  stringT=malloc(sizeof(char));
                  strncpy(stringT,&string[ind],1);
                  lista=aggiungi_elemento(lista,stringT);
                  stringT=NULL;
            }
      }
      lista = aggiungi_elemento(lista,end);
      n=0;
      tmp=NULL;
}

| vet_lists {
      riga++;
      controllo_dim(i,riga);
      n_colonne=i;
      i=0;
      if(tipo==4){
            int q=strlen(string);
            int ind;
            for(ind=0;ind<q;ind++){
                  char *stringT=NULL;
                  stringT=malloc(sizeof(char));
                  strncpy(stringT,&string[ind],1);
                  lista=aggiungi_elemento(lista,stringT);
                  stringT=NULL;
            }
      }
      lista = aggiungi_elemento(lista,end);
      n=0;
      tmp=NULL;
}

;

vet_lists:vet_lists VIRGOLA {printf(", ");} vet_list {
      if(tipo==4){
            i=i+n_colonne;
      }else{
            i++;
      }
      if(type!=0){
            tipo=controllo_type(type,tipo,tmp,var,l,n,riga);
      }
      type=tipo;
```

```
}

|vet_lists {printf(" ");} vet_list {
      if(tipo==4){
            i=i+n_colonne;
      }else{
            i++;
      }
      if(type!=0){
            tipo=controllo_type(type,tipo,tmp,var,l,n,riga);
      }
      type=tipo;
}

| vet_list {
      if(tipo==4){
            i=i+n_colonne;
      }else{
            i++;
      }
      if(type!=0){
            tipo=controllo_type(type,tipo,tmp,var,l,n,riga);
      }
      type=tipo;
}

/*Gestione errori*/
|vet_lists {printf(" ");} ID {
      yyerror("\nError: The expression is not allowed.\n");
      exit(0);
}
|vet_lists VIRGOLA {printf(", ");} ID{
      yyerror("\nError: The expression is not allowed.\n");
      exit(0);
}
|ID{
      yyerror("\nError: The expression is not allowed.\n");
      exit(0);
}
/*Fine gestione errori*/

;

vet_list: type_num

|STR_CHAR {
      printf("%s",$1);
      tipo = 4;
      char tmpvar[(strlen($1)-2)];
      l = n;
      if (l!=0){
            tmp=NULL;
            tmp=(char*)malloc(sizeof(string));
            tmp=string;
      }
      var=NULL;
      var=(char*)malloc(sizeof(tmpvar));
      n = strlen($1)-2;
      n_righe=1;
      n_colonne=n;
      strncpy(var,&$1[1],n);
      string=NULL;
      string=(char*)malloc(sizeof(var));
      string=var;
```

```
      if (fr==1){
            int q=strlen(string);
            int ind;
            for(ind=0;ind<q;ind++){
                  char *stringT=NULL;
                  stringT=malloc(sizeof(char));
                  strncpy(stringT,&string[ind],1);
                  lista=aggiungi_elemento(lista,stringT);
                  stringT=NULL;
            }
            lista = aggiungi_elemento(lista,end);
      }
}

;

type_num:PIU UNSIGNED_NUMBER{

      printf("+%d",$2);
      tipo = 1;
      n_righe=1;
      n_colonne=1;
      string=NULL;
      string = (char*)malloc(sizeof($2));
      sprintf(string, "%d", $2);
      lista = aggiungi_elemento(lista,string);
}

|MENO UNSIGNED_NUMBER{

      printf("-%d",$2);
      tipo = 1;
      n_righe=1;
      n_colonne=1;
      string=NULL;
      string = (char*)malloc(sizeof($2)+1);
      sprintf(string, "%d", -$2);
      lista = aggiungi_elemento(lista,string);
}

|MENO FLOAT_NUMBER{

      printf("-%f",$2);
      tipo = 3;
      n_righe=1;
      n_colonne=1;
      string=NULL;
      string = (char*)malloc(sizeof($2)+1);
      sprintf(string, "%f", -$2);
      lista = aggiungi_elemento(lista,string);
}
|PIU FLOAT_NUMBER{

      printf("+%f",$2);
      tipo = 3;
      n_righe=1;
      n_colonne=1;
      string=NULL;
      string = (char*)malloc(sizeof($2));
      sprintf(string, "%f", $2);
      lista = aggiungi_elemento(lista,string);
}
|UNSIGNED_NUMBER{
```

```
      printf("%d",$1);
      tipo = 1;
      n_righe=1;
      n_colonne=1;
      string=NULL;
      string = (char*)malloc(sizeof($1));
      sprintf(string, "%d", $1);
      lista = aggiungi_elemento(lista,string);
}
|FLOAT_NUMBER{

      printf("%f",$1);
      tipo = 3;
      n_righe=1;
      n_colonne=1;
      string=NULL;
      string = (char*)malloc(sizeof($1));
      sprintf(string, "%f", $1);
      lista = aggiungi_elemento(lista,string);
}

;


op_arit : op_arit {

            typeA=tipo;

            /*SE UNO DEI DUE FATTORI è UNA STRINGA NON VA AVANTI*/
            /*Gestione errori*/
            if(tipo==4||typeA==4){
                  yyerror("\nError: The expression is not allowed.\n");
                  exit(0);
            }
            /*Fine gestione errori*/

            vcolonneA=n_colonne;
            vrigheA=n_righe;

} PIU {printf(" + ");fr=0;} term{

      /*SE UNO DEI DUE FATTORI è UNA STRINGA NON VA AVANTI*/
      /*Gestione errori*/
      if(tipo==4||typeA==4){
            yyerror("\nError: The expression is not allowed.\n");
            exit(0);
      }
      /*Fine gestione errori*/

      if(n_colonne==n_righe&&n_righe==1){
            if (tipo==1){
                  opBi = op1i;
            }
            if(tipo==3){
                  opBf = op1f;
            }
      }
      else{
            opB=NULL;
            opB=(struct elementoLista *)malloc(sizeof(op1));
            opB= op1;
```

```
	}

	/*SOMMA TRA DUE SCALARI*/
	if(vcolonneA==n_colonne&&vrigheA==n_righe&&vcolonneA==1&&vrigheA==1
){
		/*I DUE SCALARI SONO INTERI*/
		if(tipo==1&&typeA==1){

			opAi = opAi + opBi;
			string=NULL;
			string = (char*)malloc(sizeof(opAi));
			sprintf(string,"%d",opAi);
			lista=NULL;
			lista = aggiungi_elemento(lista,string);

		}
		else{
			if(typeA==1&&tipo==3){
				opAf = (float )opAi + opBf;
			}
			if(typeA==3&&tipo==1)
				opAf = opAf + (float )opBi;
			if(typeA==3&&tipo==3)
				opAf = opAf + opBf;
			string=NULL;
			string = (char*)malloc(sizeof(opAf));
			sprintf(string,"%f",opAf);
			lista=NULL;
			lista = aggiungi_elemento(lista,string);
			tipo=3;
		}
	}
	else{
		/*SOMMA DI UNA MATRICE(VETTORE)PER SCALARE*/
		if(n_righe==n_colonne&&n_colonne==1){
			if(vrigheA!=0&&vcolonneA!=0){
				if(tipo==1&&typeA==1){
					struct elementoLista *opAt=NULL;
					opAt=somma_scalare_vettore(opA, opBi,
opBf,tipo,typeA,n_colonne,n_righe,vcolonneA,vrigheA);
					opA=NULL;
					opA=(struct elementoLista
*)malloc(sizeof(opAt));
					opA=opAt;
				}
				else{
					struct elementoLista *opAt=NULL;
					opAt=somma_scalare_vettore(opA, opBi,
opBf,tipo,typeA,n_colonne,n_righe,vcolonneA,vrigheA);
					opA=NULL;
					opA=(struct elementoLista
*)malloc(sizeof(opAt));
					opA=opAt;
					tipo=3;
				}
				lista=NULL;
				lista=(struct elementoLista
*)malloc(sizeof(opA));
				lista=opA;
			}
			else{
				lista=NULL;
				lista=aggiungi_elemento(lista,"NULL");
				tipo=0;
```

```
                        opA=NULL;
                        opA=(struct elementoLista
*)malloc(sizeof(lista));
                        opA=lista;
                  }
                  n_colonne=vcolonneA;
                  n_righe=vrigheA;
                  struct elementoLista *opAt=NULL;
                  opAt=riposizionamento(opA,n_righe,n_colonne);
                  opA=NULL;
                  opA=(struct elementoLista *)malloc(sizeof(opAt));
                  opA=opAt;
            }
            else{
                  /*SOMMA DI UNO SCALARE PER UNA MATRICE(VETTORE)*/
                  if(vrigheA==vcolonneA&&vcolonneA==1){
                        if(n_righe!=0&&n_colonne!=0){
                              if(tipo==1&&typeA==1){
                                    struct elementoLista *opAt=NULL;
                                    opAt=somma_scalare_vettore(opB, opAi,
opAf,tipo,typeA,n_colonne,n_righe,vcolonneA,vrigheA);
                                    opA=NULL;
                                    opA=(struct elementoLista
*)malloc(sizeof(opAt));
                                    opA=opAt;
                              }
                              else{
                                    struct elementoLista *opAt=NULL;
                                    opAt=somma_scalare_vettore(opB, opAi,
opAf,tipo,typeA,n_colonne,n_righe,vcolonneA,vrigheA);
                                    opA=NULL;
                                    opA=(struct elementoLista
*)malloc(sizeof(opAt));
                                    opA=opAt;
                                    tipo=3;
                              }
                              lista=NULL;
                              lista=(struct elementoLista
*)malloc(sizeof(opA));
                              lista=opA;
                        }
                        else{
                              lista=NULL;
                              lista=aggiungi_elemento(lista,"NULL");
                              tipo=0;
                              opA=NULL;
                              opA=(struct elementoLista
*)malloc(sizeof(lista));
                              opA=lista;
                        }
                        struct elementoLista *opAt=NULL;
                        opAt=riposizionamento(opA,n_righe,n_colonne);
                        opA=NULL;
                        opA=(struct elementoLista *)malloc(sizeof(opAt));
                        opA=opAt;
                  }
                  else{
                        /*SOMMA TRA MATRICI(VETTORI)*/

      if(n_righe!=0&&n_colonne!=0&&vrigheA!=0&&vcolonneA!=0){
                              if(tipo==1&&typeA==1){
                                    struct elementoLista *opAt=NULL;

      opAt=somma(opA,opB,tipo,typeA,n_colonne,n_righe,vcolonneA,vrigheA);
```

```
                                    opA=NULL;
                                    opA=(struct elementoLista
*)malloc(sizeof(opAt));
                                    opA=opAt;
                              }
                              else{
                                    struct elementoLista *opAt=NULL;

      opAt=somma(opA,opB,tipo,typeA,n_colonne,n_righe,vcolonneA,vrigheA);
                                    opA=NULL;
                                    opA=(struct elementoLista
*)malloc(sizeof(opAt));
                                    opA=opAt;
                                    tipo=3;
                              }
                              lista=NULL;
                              lista=(struct elementoLista
*)malloc(sizeof(opA));
                              lista=opA;


                        }
                        else{

      if(n_righe==0&&n_colonne==0&&vrigheA==0&&vcolonneA==0){
                                    lista=NULL;

      lista=aggiungi_elemento(lista,"NULL");
                                    tipo=0;
                                    n_colonne=0;
                                    n_righe=0;
                                    opA=NULL;
                                    opA=(struct elementoLista
*)malloc(sizeof(lista));
                                    opA=lista;
                              }
                              else{
                                    yyerror("\nError using plus: Matrix
dimensions must agree.\n");
                                    exit(0);
                              }
                        }

                        struct elementoLista *opAt=NULL;
                        opAt=riposizionamento(opA,n_righe,n_colonne);
                        opA=NULL;
                        opA=(struct elementoLista *)malloc(sizeof(opAt));
                        opA=opAt;

                  }
            }
      }
}

|op_arit {

            typeA=tipo;

            /*SE UNO DEI DUE FATTORI è UNA STRINGA NON VA AVANTI*/
            /*Gestione errori*/
            if(tipo==4||typeA==4){
                  yyerror("\nError: The expression is not allowed.\n");
                  exit(0);
            }
```

```
            /*Fine gestione errori*/

            vcolonneA=n_colonne;
            vrigheA=n_righe;

}MENO {printf(" - ");fr=0;} term{

      if(n_colonne==n_righe&&n_righe==1){
            if (tipo==1){
                  opBi = op1i;

            }
            if(tipo==3){
                  opBf = op1f;

            }
      }
      else{
            opB=NULL;
            opB=(struct elementoLista *)malloc(sizeof(op1));
            opB= op1;
      }

      /*SE UNO DEI DUE FATTORI è UNA STRINGA NON VA AVANTI*/
      /*Gestione errori*/
      if(tipo==4||typeA==4){
            yyerror("\nError: The expression is not allowed.\n");
            exit(0);
      }
      /*Fine gestione errori*/

      /*DIFF TRA DUE SCALARI*/
      if(vcolonneA==n_colonne&&vrigheA==n_righe&&vcolonneA==1&&vrigheA==1
){
            /*I DUE SCALARI SONO INTERI*/
            if(tipo==1&&typeA==1){
                  opAi = opAi - opBi;

                  string=NULL;
                  string = (char*)malloc(sizeof(opAi));
                  sprintf(string,"%d",opAi);
                  lista=NULL;
                  lista = aggiungi_elemento(lista,string);
            }
            else{
                  if(typeA==1&&tipo==3){
                        opAf = (float )opAi - opBf;

                  }
                  if(typeA==3&&tipo==1){
                        opAf = opAf - (float )opBi;

                  }
                  if(typeA==3&&tipo==3){
                        opAf = opAf - opBf;

                  }
                  string=NULL;
                  string = (char*)malloc(sizeof(opAf));
                  sprintf(string,"%f",opAf);
                  lista=NULL;
                  lista = aggiungi_elemento(lista,string);
                  tipo=3;
            }
```

```
	}
	else{
		/*DIFFERENZA DI UNA MATRICE(VETTORE)PER SCALARE*/
		if(n_righe==n_colonne&&n_colonne==1){
			if(vrigheA!=0&&vcolonneA!=0){
				if(tipo==1&&typeA==1){
					struct elementoLista *opAt=NULL;
					opAt=differenza_scalare_vettore(opA, opBi,
opBf,tipo,typeA,n_colonne,n_righe,vcolonneA,vrigheA);
					opA=NULL;
					opA=(struct elementoLista
*)malloc(sizeof(opAt));
					opA=opAt;
				}
				else{
					struct elementoLista *opAt=NULL;
					opAt=differenza_scalare_vettore(opA, opBi,
opBf,tipo,typeA,n_colonne,n_righe,vcolonneA,vrigheA);
					opA=NULL;
					opA=(struct elementoLista
*)malloc(sizeof(opAt));
					opA=opAt;
					tipo=3;
				}
				lista=NULL;
				lista=(struct elementoLista
*)malloc(sizeof(opA));
				lista=opA;
			}
			else{
				lista=NULL;
				lista=aggiungi_elemento(lista,"NULL");
				tipo=0;
				opA=NULL;
				opA=(struct elementoLista
*)malloc(sizeof(lista));
				opA=lista;
			}
			n_colonne=vcolonneA;
			n_righe=vrigheA;
			struct elementoLista *opAt=NULL;
			opAt=riposizionamento(opA,n_righe,n_colonne);
			opA=NULL;
			opA=(struct elementoLista *)malloc(sizeof(opAt));
			opA=opAt;
		}
		else{
			/*DIFFERENZA DI UNO SCALARE PER UNA MATRICE(VETTORE)*/
			if(vrigheA==vcolonneA&&vcolonneA==1){
				if(n_righe!=0&&n_colonne!=0){
					if(tipo==1&&typeA==1){
						struct elementoLista *opAt=NULL;
						opAt=differenza_scalare_vettore(opB,
opAi, opAf,tipo,typeA,n_colonne,n_righe,vcolonneA,vrigheA);
						opA=NULL;
						opA=(struct elementoLista
*)malloc(sizeof(opAt));
						opA=opAt;
					}
					else{
						struct elementoLista *opAt=NULL;
						opAt=differenza_scalare_vettore(opB,
opAi, opAf,tipo,typeA,n_colonne,n_righe,vcolonneA,vrigheA);
						opA=NULL;
```

```
                                    opA=(struct elementoLista
*)malloc(sizeof(opAt));
                                    opA=opAt;
                                    tipo=3;
                              }
                              lista=NULL;
                              lista=(struct elementoLista
*)malloc(sizeof(opA));
                              lista=opA;
                        }
                        else{
                              lista=NULL;
                              lista=aggiungi_elemento(lista,"NULL");
                              tipo=0;
                              opA=NULL;
                              opA=(struct elementoLista
*)malloc(sizeof(lista));
                              opA=lista;
                        }
                        struct elementoLista *opAt=NULL;
                        opAt=riposizionamento(opA,n_righe,n_colonne);
                        opA=NULL;
                        opA=(struct elementoLista *)malloc(sizeof(opAt));
                        opA=opAt;
                  }
                  else{
                        /*DIFFERENZA TRA MATRICI(VETTORI)*/

      if(n_righe!=0&&n_colonne!=0&&vrigheA!=0&&vcolonneA!=0){
                              if(tipo==1&&typeA==1){
                                    struct elementoLista *opAt=NULL;

      opAt=differenza(opA,opB,tipo,typeA,n_colonne,n_righe,vcolonneA,vrig
heA);
                                    opA=NULL;
                                    opA=(struct elementoLista
*)malloc(sizeof(opAt));
                                    opA=opAt;
                              }
                              else{

                                    struct elementoLista *opAt=NULL;

      opAt=differenza(opA,opB,tipo,typeA,n_colonne,n_righe,vcolonneA,vrig
heA);
                                    opA=NULL;
                                    opA=(struct elementoLista
*)malloc(sizeof(opAt));
                                    opA=opAt;
                                    tipo=3;
                              }
                              lista=NULL;
                              lista=(struct elementoLista
*)malloc(sizeof(opA));
                              lista=opA;
                        }
                        else{

      if(n_righe==0&&n_colonne==0&&vrigheA==0&&vcolonneA==0){
                              lista=NULL;

      lista=aggiungi_elemento(lista,"NULL");
                              tipo=0;
                              n_colonne=0;
```

```
                                    n_righe=0;
                                    opA=NULL;
                                    opA=(struct elementoLista
*)malloc(sizeof(lista));
                                    opA=lista;
                              }
                              else{
                                    yyerror("\nError using minus: Matrix
dimensions must agree.\n");
                                    exit(0);
                              }
                        }
                        struct elementoLista *opAt=NULL;
                        opAt=riposizionamento(opA,n_righe,n_colonne);
                        opA=NULL;
                        opA=(struct elementoLista *)malloc(sizeof(opAt));
                        opA=opAt;
                  }
            }
      }
}

|term{
      if(n_colonne==n_righe&&n_righe==1){
            if (tipo==1){
                  opAi = op1i;
            }
            if(tipo==3){
                  opAf = op1f;

            }
      }
      else{
            opA=NULL;
            opA=(struct elementoLista *)malloc(sizeof(op1));
            opA = op1;
      }
      vcolonneA=n_colonne;
      vrigheA=n_righe;

}

;

term: term{

      type=tipo;

      /*SE UNO DEI DUE FATTORI è UNA STRINGA NON VA AVANTI*/
      /*Gestione errori*/
      if(tipo==4||type==4){
            yyerror("\nError: The expression is not allowed.\n");
            exit(0);
      }
      /*Fine gestione errori*/

      vcolonne1=n_colonne;
      vrighe1=n_righe;

} PER {printf(" * ");fr=0;} operando{

      if(n_colonne==n_righe&&n_righe==1){
            if (tipo==1){
                  op2i = opi;
```

```
		}
		if(tipo==3){
			op2f = opf;
		}
	}
	else{
		op2=NULL;
		op2=(struct elementoLista *)malloc(sizeof(op));
		op2= op;
	}

	/*SE UNO DEI DUE FATTORI è UNA STRINGA NON VA AVANTI*/
	/*Gestione errori*/
	if(tipo==4||type==4){
		yyerror("\nError: The expression is not allowed.\n");
		exit(0);
	}
	/*Fine gestione errori*/

	/*PRODOTTO TRA DUE SCALARI*/
	if(vcolonne1==n_colonne&&vrighe1==n_righe&&vcolonne1==1&&vrighe1==1
){
		if(type==1&&tipo==1){
			op1i = op1i * op2i;
			string=NULL;
			string = (char*)malloc(sizeof(op1i));
			sprintf(string,"%d",op1i);
			lista=NULL;
			lista = aggiungi_elemento(lista,string);
		}
		else{
			if(type==1&&tipo==3){
				op1f = (float )op1i * op2f;
			}
			if(type==3&&tipo==1){
				op1f = op1f * (float )op2i;
			}
			if(type==3&&tipo==3){
				op1f = op1f * op2f;
			}
			string=NULL;
			string = (char*)malloc(sizeof(op1f));
			sprintf(string,"%f",op1f);
			lista=NULL;
			lista = aggiungi_elemento(lista,string);
			tipo=3;
		}
	}
	else{
		/*PRODOTTO DI UNA MATRICE(VETTORE)PER SCALARE*/
		if(n_righe==n_colonne&&n_colonne==1){
			if(vrighe1!=0&&vcolonne1!=0){
				if(tipo==1&&type==1){
					struct elementoLista *op1t=NULL;
					op1t=moltiplicazione_scalare_vettore(op1,
op2i, op2f,tipo,type,n_colonne,n_righe,vcolonne1,vrighe1);
					op1=NULL;
					op1=(struct elementoLista
*)malloc(sizeof(op1t));
					op1=op1t;
				}
				else{
					struct elementoLista *op1t=NULL;
```

```
                              op1t=moltiplicazione_scalare_vettore(op1,
op2i, op2f,tipo,type,n_colonne,n_righe,vcolonne1,vrighe1);
                              op1=NULL;
                              op1=(struct elementoLista
*)malloc(sizeof(op1t));
                              op1=op1t;
                              tipo=3;
                        }
                        lista=NULL;
                        lista=(struct elementoLista
*)malloc(sizeof(op1));
                        lista=op1;
                  }
                  else{
                        lista=NULL;
                        lista=aggiungi_elemento(lista,"NULL");
                        tipo=0;
                        op1=NULL;
                        op1=(struct elementoLista
*)malloc(sizeof(lista));
                        op1=lista;
                  }
                  n_colonne=vcolonne1;
                  n_righe=vrighe1;
                  struct elementoLista *op1t=NULL;
                  op1t=riposizionamento(op1,n_righe,n_colonne);
                  op1=NULL;
                  op1=(struct elementoLista *)malloc(sizeof(op1t));
                  op1=op1t;
            }
            else{
                  /*PRODOTTO DI UNO SCALARE PER UNA MATRICE(VETTORE)*/
                  if(vrighe1==vcolonne1&&vcolonne1==1){
                        if(n_righe!=0&&n_colonne!=0){
                              if(tipo==1&&type==1){
                                    struct elementoLista *op1t=NULL;

      op1t=moltiplicazione_scalare_vettore(op2, op1i,
op1f,tipo,type,n_colonne,n_righe,vcolonne1,vrighe1);
                                    op1=NULL;
                                    op1=(struct elementoLista
*)malloc(sizeof(op1t));
                                    op1=op1t;
                              }
                              else{
                                    struct elementoLista *op1t=NULL;

      op1t=moltiplicazione_scalare_vettore(op2, op1i,
op1f,tipo,type,n_colonne,n_righe,vcolonne1,vrighe1);
                                    op1=NULL;
                                    op1=(struct elementoLista
*)malloc(sizeof(op1t));
                                    op1=op1t;
                                    tipo=3;
                              }
                              lista=NULL;
                              lista=(struct elementoLista
*)malloc(sizeof(op1));
                              lista=op1;
                        }
                        else{
                              lista=NULL;
                              lista=aggiungi_elemento(lista,"NULL");
                              tipo=0;
```

```
                              op1=NULL;
                              op1=(struct elementoLista
*)malloc(sizeof(lista));
                              op1=lista;
                        }
                        struct elementoLista *op1t=NULL;
                        op1t=riposizionamento(op1,n_righe,n_colonne);
                        op1=NULL;
                        op1=(struct elementoLista *)malloc(sizeof(op1t));
                        op1=op1t;
                  }
                  else{
                        /*PRODOTTO RIGHA PER COLONNA*/

      if(n_righe!=0&&n_colonne!=0&&vrighe1!=0&&vcolonne1!=0){
                              if(tipo==1&&type==1){
                                    struct elementoLista *op1t=NULL;

      op1t=moltiplicazione_rigacolonna(op1,op2,vcolonne1,n_righe,n_colonn
e,tipo,type);
                                    op1=NULL;
                                    op1=(struct elementoLista
*)malloc(sizeof(op1t));
                                    op1=op1t;
                              }
                              else{

                                    struct elementoLista *op1t=NULL;

      op1t=moltiplicazione_rigacolonna(op1,op2,vcolonne1,n_righe,n_colonn
e,tipo,type);
                                    op1=NULL;
                                    op1=(struct elementoLista
*)malloc(sizeof(op1t));
                                    op1=op1t;
                                    tipo=3;
                              }
                              lista=NULL;
                              lista=(struct elementoLista
*)malloc(sizeof(op1));
                              lista=op1;


                              n_righe=vrighe1;
                        }
                        else{
                              lista=NULL;
                              lista=aggiungi_elemento(lista,"NULL");
                              tipo=0;
                              n_colonne=0;
                              n_righe=0;
                              op1=NULL;
                              op1=(struct elementoLista
*)malloc(sizeof(lista));
                              op1=lista;
                        }
                        struct elementoLista *op1t=NULL;
                        op1t=riposizionamento(op1,n_righe,n_colonne);
                        op1=NULL;
                        op1=(struct elementoLista *)malloc(sizeof(op1t));
                        op1=op1t;
                  }
            }
      }
```

```
}

|term { type=tipo;

      /*SE UNO DEI DUE TERMINI è UNA STRINGA NON VA AVANTI*/
      /*Gestione errori*/
      if(tipo==4){
            yyerror("\nError: The expression is not allowed.\n");
            exit(0);
      }
      /*Fine gestione errori*/

      vcolonne1=n_colonne;
      vrighe1=n_righe;

}DIVISO {printf(" / ");fr=0;} operando{

      if(n_colonne==n_righe&&n_righe==1){
            if (tipo==1){
                  op2i = opi;
            }
            if(tipo==3){
                  op2f = opf;
            }
      }
      else{
            yyerror("\nError: The expression is not allowed.\n");
            exit(0);
      }


      /*SE UNO DEI DUE TERMINI è UNA STRINGA NON VA AVANTI*/
      /*Gestione errori*/
      if(tipo==4){
            yyerror("\nError: The expression is not allowed.\n");
            exit(0);
      }
      if(tipo==1){
            if(op2i==0){
                  yyerror("\nError: The expression is not allowed.\n");
                  exit(0);
            }
      }
      if(tipo==3){
            if(op2f==0){
                  yyerror("\nError: The expression is not allowed.\n");
                  exit(0);
            }
      }
      /*Fine gestione errori*/

      /*DIvISIONE TRA DUE SCALARI*/
      if(vcolonne1==n_colonne&&vrighe1==n_righe&&vcolonne1==1&&vrighe1==1
){

            if(type==1&&tipo==1){
                  op1f = (float )op1i / (float )op2i;
            }
            if(type==1&&tipo==3){
                  op1f = (float )op1i / op2f;
            }
            if(type==3&&tipo==1){
                  op1f = op1f / (float )op2i;
            }
```

```
            if(type==3&&tipo==3){
                  op1f = op1f / op2f;
            }
            string=NULL;
            string = (char*)malloc(sizeof(op1f));
            sprintf(string,"%f",op1f);
            lista=NULL;
            lista = aggiungi_elemento(lista,string);
      }
      else{
            struct elementoLista *op1t=NULL;
            op1t=divisione(op1,op2i,op2f,vrighe1, vcolonne1,tipo);
            op1=NULL;
            op1=(struct elementoLista *)malloc(sizeof(op1t));
            op1=op1t;
            n_colonne=vcolonne1;
            n_righe=vrighe1;
            lista=NULL;
            lista=(struct elementoLista *)malloc(sizeof(op1));
            lista=op1;
            op1t=NULL;
            op1t=riposizionamento(op1,n_righe,n_colonne);
            op1=NULL;
            op1=(struct elementoLista *)malloc(sizeof(op1t));
            op1=op1t;
      }
      type=tipo;
      tipo=3;
}

|operando{

      if(n_colonne==n_righe&&n_righe==1){
            if (tipo==1){
                  op1i = opi;
            }
            if(tipo==3){
                  op1f = opf;
            }
      }
      else{
            op1=NULL;
            op1=(struct elementoLista *)malloc(sizeof(op));
            op1 = op;
      }
      vcolonne1=n_colonne;
      vrighe1=n_righe;
}

;

operando: vet_list {
      if (tipo==1){
            opi = atoi(string);
      }
      if (tipo==3){
            opf = atof(string);
      }
}

| operando APICE {
      printf("' ");

      if(n_righe==1&&n_colonne==1){
```

```
            lista=NULL;
            if(tipo==1){
                  string=NULL;
                  string = (char*)malloc(sizeof(opi));
                  sprintf(string,"%d",opi);
                  lista=aggiungi_elemento(lista,string);
            }
            if(tipo==3){
                  string=NULL;
                  string = (char*)malloc(sizeof(opf));
                  sprintf(string,"%f",opf);
                  lista=aggiungi_elemento(lista,string);
            }
      }
      else{
            struct elementoLista *opt=NULL;
            opt=trasposta(op,n_righe,n_colonne);
            op=NULL;
            op=(struct elementoLista *)malloc(sizeof(opt));
            op=opt;
            lista=NULL;
            lista=(struct elementoLista *)malloc(sizeof(op));
            lista=op;
            opt=NULL;
            opt=riposizionamento(op,n_righe,n_colonne);
            op=NULL;
            op=(struct elementoLista *)malloc(sizeof(opt));
            op=opt;
      }

      vcolonne=n_colonne;
      n_colonne=n_righe;
      n_righe=vcolonne;
}

|ID{
      printf("%s",$1);
      s=NULL;
      s=(symrec *)malloc(sizeof(symrec));
      s=getsym ($1);
      if (s==0){
            yyerror("\nError: Undefined function or variable.\n");
            exit(0);
      }
      else{
            tipo=s->type;
            n_righe = s->n_righe;
            n_colonne = s->n_colonne;
            h=NULL;
            h =(struct elementoLista *)malloc(sizeof(s->lista));
            h = s->lista;
            if (n_righe==n_colonne&&n_righe==1){
                  if(tipo==1||tipo==3)
                  {
                        string=NULL;
                        string = (char*)malloc(sizeof(h->string));
                        strcpy(string,h->string);
                        lista=NULL;
                        lista=aggiungi_elemento(lista,string);
                        string=NULL;
                  }
                  else{
                        /*HO UNA STRINGA DI UN SOLO CARATTERE*/
                        h=h->pun;//serve a scartare end
```

```
                        string=NULL;
                        string = (char*)malloc(sizeof(h->string));
                        strcpy(string,h->string);
                        lista=NULL;
                        lista=aggiungi_elemento(lista,string);
                        string=NULL;
                  }
            }
            else{
                  if(n_righe!=0&&n_colonne!=0){
                        lista=NULL;
                        ris=NULL;
                        ris=(struct elementoLista *)malloc(sizeof(s-
>lista));
                        while(h!=NULL){
                              ris=h->punBack;
                              h=h->pun;
                        }
                        if(ris!=NULL){
                              h=ris->pun;
                        }
                        while(h!=NULL){
                              string=NULL;
                              string = (char*)malloc(sizeof(h->string));
                              strcpy(string,h->string);
                              lista=aggiungi_elemento(lista,string);
                              h=h->punBack;
                              string=NULL;
                        }
                  }
                  else{
                        lista=NULL;
                        lista=aggiungi_elemento(lista,"NULL");
                  }
            }

            if(n_colonne==n_righe&&n_righe==0){}
            else{
                  if(n_colonne==n_righe&&n_righe==1){
                        if(tipo==1){
                              opi=atoi(s->lista->string);
                        }
                        if(tipo==3){
                              opf=atof(s->lista->string);
                        }
                  }
                  else{
                        op=NULL;
                        op =(struct elementoLista *)malloc(sizeof(s-
>lista));
                        op = ris->pun;


                  }
            }
      }
}

;


%%
```

```
int main( int argc, char *argv[] )
{
      extern FILE *yyin;

      if(argc!=2){
            printf("Errore!\nPer far partire il programma serve il
nome\ndel file .m che si vuole analizzare\n");
            printf("ESEMPIO: %s <file_da_analizzare> \n",argv[0]);
            return 0;
      }
      --argc;
      ++argv;

      if((yyin = fopen( argv[0], "r" ))==NULL){
            printf("Errore nell'apertura del file, controllare\n");
            printf("che la path inserita (%s) sia corretta\n", argv[0]);
            return 0;
      }

      yyparse ();

      return 0;

}

void yyerror ( char *s ) /* chiamata da yyparse in caso di errore */
{
            printf ("%s\n", s);
}

/*controllo sui tipi*/

int controllo_type(int type,int tipo,char *tmp,char *var,int l,int n,int
riga)
{
      if (type==tipo){
            if(tipo==4){
                  if(l!=0){
                        char *temp;
                        temp = (char*)malloc(l*sizeof(char));
                        temp=tmp;
                        strncat(temp,var,n);
                        string=NULL;
                        string=(char *)malloc(sizeof(temp));
                        string=temp;
                        d = strlen(temp);
                        temp=NULL;
                  }
            }
      }
      else
      {
            if(tipo==4){
                  printf("\n\nWarning: Out of range or non-integer values
truncated during conversion to character.");
                  exit(0);
            }
            else{
                  tipo=3;
            }
      }
      return(tipo);
}
```

semantica.h

```
/*Definiamo la variabile tipo_var, la quale può assumere uno dei
qualsiasi valori indicati tra graffe*/

enum tipo_var {INT=1,FLOAT=3,CHAR=4,null=0};

/*Con typedef creiamo un nuovo tipo di dato, tipo_variabile, di tipo
tipo_var*/

typedef enum tipo_var tipo_variabile;

int nuovo;
int vecchio;

struct elementoLista {
      char *string;
      struct elementoLista *pun;
      struct elementoLista *punBack;
};

/*Prototipi delle funzioni*/
struct elementoLista *aggiungi_elemento(struct elementoLista *lista, char
*string);

void stampa(struct elementoLista *lista);


/*----------------RECORD DELLA TABELLA DEI SIMBOLI-----------------*/

/*Un record della tabella dei simboli è una struttura avente i seguenti
campi:*/

struct symrec
{
      char *name; /*nome del simbolo*/
      struct symrec *next;    /*puntatore a un campo della struttura*/
      tipo_variabile type;    /*tipo della variabile*/
      struct elementoLista *lista;  /*valori associati al simbolo*/
      int n_righe;
      int n_colonne;
};

typedef struct symrec symrec;

/*sym_table è un puntatore alla struttura symrec (la tabella dei simboli)
e lo inizializziamo a zero.*/

symrec *sym_table = (symrec *)0;


/*Le operazioni possibili sulla symbol table sono di tipo put e get*/

/*Con la funzione putsym restituiamo il puntatore alla tabella dei
simboli, che punta al nuovo record inserito. Il nuovo simbolo viene
inserito in testa, dopo che gli è stata allocata la giusta memoria per
contenerlo.*/

symrec * putsym (char *sym_name, int val,struct elementoLista *lista, int
n_righe, int n_colonne)
{
      symrec *ptr;
      ptr = (symrec *) malloc (sizeof(symrec));
```

```
      ptr->name = (char *) malloc (strlen(sym_name)+1);
      strcpy (ptr->name,sym_name);
      ptr->type=val;
      ptr->lista=NULL;
      ptr->lista= (struct elementoLista *)malloc(sizeof(lista));
      ptr->lista=lista;
      ptr->next = (struct symrec *)sym_table;
      ptr->n_righe=n_righe;
      ptr->n_colonne=n_colonne;
      sym_table = ptr;
      return(ptr);
}

/*Con la funzione putsym2 restituiamo il puntatore alla tabella dei
simboli, dopo aver sovrascritto il valore in corrispondenza di un simbolo
già presente nella symbol table.*/

void putsym2 (char *sym_name, int val,symrec *ptr, struct elementoLista
*lista, int n_righe, int n_colonne)
{
      ptr->type=val;
      ptr->lista=NULL;
      ptr->lista= (struct elementoLista *)malloc(sizeof(lista));
      ptr->lista=lista;
      ptr->n_righe=n_righe;
      ptr->n_colonne=n_colonne;
      return;
}

/*Con la funzione getsym restituiamo un puntatore alla tabella dei
simboli. Come argomento viene passato il simbolo che si vuole prelevare
dalla tabella dei simboli. Viene scansionata tutta la tabella, finchè il
simbolo cercato non è uguale al simbolo che risiede nel record i-esimo
della tabella dei simboli. Se trovato, viene restituito il puntatore al
simbolo e viene stampato su terminale il record corrispondente*/

symrec * getsym (char *sym_name)
{
      symrec *ptr;
      for (ptr = sym_table; ptr!=(symrec *)0; ptr=(symrec *)ptr->next)
            if (strcmp (ptr->name,sym_name) == 0){
                  printf("\nSYMBOL
TABLE:\tname:%s\ttype:%d\tr:%d\tc:%d\t",ptr->name,ptr->type,ptr-
>n_righe,ptr->n_colonne);
                  printf("\nLISTA VALORI\t");
                  stampa(ptr->lista);
                  return ptr;
            }
      return 0;
}


/*Con la funzione install effettuiamo un controllo semantico. L'obiettivo
della funzione è guidare l'inserimento di un nuovo record nella symbol
table. Passiamo, come parametri, il puntatore al simbolo appena immesso,
il suo tipo, la lista di valori e la sua dimensione in termini di righe e
colonne. Creiamo un puntatore alla tabella dei simboli e mettiamo in tale
puntatore l'eventuale puntatore corrispondente al nuovo simbolo immesso.
Se tale puntatore non viene trovato, significa che il simbolo non è
presente
nella tabella dei simboli. In questo caso, tramite la funzione putsym,
inseriamo questo simbolo nella tabella. Se invece viene trovato,
significa che abbiamo già inserito il nome di quel simbolo nella tabella
e il record viene sovrascritto mediante la funzione putsym2*/
```

```
void install (char *sym_name,int tipo_var,struct elementoLista *lista,
int n_righe, int n_colonne)
{
      symrec *s;
      s = getsym (sym_name);
      if (s == 0){
            s = putsym(sym_name,tipo_var,lista,n_righe,n_colonne);
      }
      else {
            putsym2 (sym_name,tipo_var,s,lista,n_righe,n_colonne);
            printf( "La variabile \"%s\" e\' stata ridefinita\r\n",
sym_name );
      }

}

/*Con la funzione clear viene svuotata la tabella dei simboli a partire
dal record di testa*/

void clear()
{
      symrec *ptr,*next;
      for ( ptr = sym_table; ptr!=(symrec *)0; )
      {
            next=ptr;
            ptr=(symrec *)ptr->next;
            free(next);
      }
      sym_table = (symrec *)0;
}


/*Controllo dimensione*/

void controllo_dim(int i,int riga)
{
      if (riga==1){
            vecchio = i;
            nuovo = i;
      }
      else
      {
            nuovo = i;
            if (nuovo == vecchio){
                  vecchio = nuovo;
            }else{
                  printf("\n\nError using vertcat: CAT arguments
dimensions are not consistent.");
                  exit(0);
            }
      }
}

/*Aggiungi elemento lista*/

struct elementoLista *aggiungi_elemento(struct elementoLista *lista, char
*string)
{
      struct elementoLista *punt;
      if(lista != NULL)
      {
            punt = (struct elementoLista *)malloc(sizeof(struct
```

```
elementoLista));
            punt->string=NULL;
            punt->string=(char *)malloc(sizeof(string));
            punt->string=string;
            punt->pun = lista;
            punt->punBack=NULL;
            lista->punBack=punt;
      } else {
            /* creazione primo elemento */
            lista = (struct elementoLista *)malloc(sizeof(struct
elementoLista));
            lista->string=NULL;
            lista->string=(char *)malloc(sizeof(string));
            lista->string=string;
            lista->pun=NULL;
            lista->punBack=NULL;
            punt=lista;
      }
      return (punt);
}

/*stampa lista*/

void stampa(struct elementoLista *lista)
{
      // Stampa la lista
      while (lista != NULL)
      {
            printf("%s\t", lista->string);
            // legge l'elemento successivo
            lista= lista->pun;
      }
      return;
}

struct elementoLista *trasposta(struct elementoLista *op,int n_righe, int
n_colonne){

      struct elementoLista *lista = NULL;
      struct elementoLista *h = NULL;
      struct elementoLista *ris=NULL;




      lista =(struct elementoLista *)malloc(sizeof(struct elementoLista
*));
      h =(struct elementoLista *)malloc(sizeof(op));
      h=op;
      struct elementoLista *hp;
      hp=NULL;
      hp =(struct elementoLista *)malloc(sizeof(op));
      lista=NULL;
      if(n_righe!=0&&n_colonne!=0){

            ris=(struct elementoLista *)malloc(sizeof(op));

            while(h!=NULL){
                  ris=h->punBack;
                  h=h->pun;
            }
            if(ris!=NULL){
                  h=ris->pun;
            }
```

```
            hp=h;
            while(strcmp(h->string,"/")!=0){
                  while(h!=NULL)
                  {
                        lista=aggiungi_elemento(lista,h->string);
                        int a;
                        for(a=1;a<=n_colonne+1;a++){
                              if(h!=NULL){
                                    h=h->punBack;
                              }
                        }
                  }
                  lista=aggiungi_elemento(lista,"/");

                  h=hp->punBack;
                  hp=h;
            }
      }
      else{
            lista=aggiungi_elemento(lista,"NULL");

      }

      return lista;

}

struct elementoLista *divisione(struct elementoLista *op1, int op2i,
float op2f,int vrighe, int vcolonne,int tipo){

      struct elementoLista *lista = NULL;
      float fnum;
      char *string=NULL;

      if(tipo==1){
            fnum=(float )op2i;
      }
      if(tipo==3){
            fnum=op2f;
      }
      lista=NULL;
      if(vrighe==0||vcolonne==0){
            lista=aggiungi_elemento(lista,"NULL");
      }
      else{
            while(op1!=NULL)
            {
                  if(strcmp(op1->string,"/")==0)
                  {
                        lista=aggiungi_elemento(lista,"/");
                  }
                  else{
                        float tmp = atof(op1->string)/fnum;
                        string=NULL;
                        string = (char*)malloc(sizeof(tmp));
                        sprintf(string, "%f", tmp);
                        lista=aggiungi_elemento(lista,string);
                  }
                  op1=op1->punBack;
            }
      }
      return lista;

}
```

```
struct elementoLista *moltiplicazione_scalare_vettore(struct
elementoLista *op1, int op2i, float op2f,int tipo,int type,int
n_colonne,int n_righe,int vcolonne, int vrighe){

	struct elementoLista *lista = NULL;
	char *string=NULL;

	if(tipo==1&&type==1){
		while(op1!=NULL){
			if(strcmp(op1->string,"/")==0)
			{
				lista=aggiungi_elemento(lista,"/");
				op1=op1->punBack;
			}
			else{
				string=NULL;
				string = (char*)malloc(sizeof(op1->string));
				strcpy(string,op1->string);
				int tmp=op2i*atoi(string);
				string=NULL;
				string = (char*)malloc(sizeof(tmp));
				sprintf(string,"%d",tmp);
				lista=aggiungi_elemento(lista,string);
				op1=op1->punBack;
			}
		}
	}
	else{
		if(n_colonne==n_righe&&n_righe==1){
			if(tipo==1)
				op2f=(float )op2i;
		}
		else{
			if(vcolonne==vrighe&&vrighe==1){
				if(type==1)
					op2f=(float )op2i;
			}
		}
		while(op1!=NULL){
			if(strcmp(op1->string,"/")==0)
			{
				lista=aggiungi_elemento(lista,"/");
				op1=op1->punBack;
			}
			else{
				string=NULL;
				string = (char*)malloc(sizeof(op1->string));
				strcpy(string,op1->string);
				float tmp=op2f*atof(string);
				string=NULL;
				string = (char*)malloc(sizeof(tmp));
				sprintf(string,"%f",tmp);
				lista=aggiungi_elemento(lista,string);
				op1=op1->punBack;
			}
		}
	}

	return lista;
}

struct elementoLista *moltiplicazione_rigacolonna(struct elementoLista
*op1, struct elementoLista *op2,int vcolonne, int n_righe, int
```

```
n_colonne,int tipo, int type){

	struct elementoLista *lista = NULL;
	char *string=NULL;

	if(vcolonne==n_righe){
		int tmpi=0;
		float tmpf=0;
		int i;
		struct elementoLista *hp;
		hp=malloc(sizeof(struct elementoLista *));
		hp=op1;
		struct elementoLista *hp1;
		hp1=malloc(sizeof(struct elementoLista *));
		hp1=op2;
		while(op1!=NULL)
		{
			for(i=1;i<=n_colonne;i++)
			{
				while(strcmp(op1->string,"/")!=0)
				{
					if(tipo==type&&tipo==1){
						tmpi=tmpi+(atoi(op1-
>string)*atoi(op2->string));
					}
					else{
						tmpf=tmpf+(atof(op1-
>string)*atof(op2->string));
					}
					op1=op1->punBack;
					int a=1;
					for(a=1;a<=n_colonne+1;a++){
						if(op2!=NULL){
							op2=op2->punBack;
						}
					}
				}
				string=NULL;
				if(tipo==type&&tipo==1){
					string = (char*)malloc(sizeof(tmpi));
					sprintf(string,"%d",tmpi);
				}
				else{
					string = (char*)malloc(sizeof(tmpf));
					sprintf(string,"%f",tmpf);
				}
				lista=aggiungi_elemento(lista,string);
				tmpi=0;
				tmpf=0;
				op1=hp;
				int a;
				op2=hp1;
				for(a=1;a<=i;a++){
					op2=op2->punBack;
				}
			}
			lista=aggiungi_elemento(lista,"/");
			int a=1;
			for(a=1;a<=vcolonne+1;a++){
				if(op1!=NULL){
					op1=op1->punBack;
				}
			}
			hp=op1;
```

```
                op2=hp1;
            }
            if(op1!=NULL){
                printf("\nError: Unexpected error.\n");
                exit(0);
            }
      }
      else{
            printf("\nError using mtimes: Inner matrix dimensions must
agree.\n");
            exit(0);
      }

      return lista;
}

struct elementoLista *riposizionamento(struct elementoLista *op,int
n_righe, int n_colonne){

      struct elementoLista *ris = NULL;

      if(n_righe>0&&n_colonne>0){
            ris=(struct elementoLista *)malloc(sizeof(op));
            while(op!=NULL){
                ris=op->punBack;
                op=op->pun;
            }
            if(ris!=NULL){
                op=ris->pun;
            }

            op = ris->pun;
      }

      return op;
}

struct elementoLista *differenza_scalare_vettore(struct elementoLista
*op1, int op2i, float op2f,int tipo,int type,int n_colonne,int
n_righe,int vcolonne, int vrighe){

      struct elementoLista *lista = NULL;
      char *string=NULL;


      if(tipo==1&&type==1){
            while(op1!=NULL){
                if(strcmp(op1->string,"/")==0)
                {
                      lista=aggiungi_elemento(lista,"/");
                      op1=op1->punBack;
                }
                else{
                      string=NULL;
                      string = (char*)malloc(sizeof(op1->string));
                      strcpy(string,op1->string);
                      int tmp=op2i-atoi(string);
                      if(n_colonne==n_righe&&n_righe==1)
                            tmp=tmp*(-1);
                      string=NULL;
                      string = (char*)malloc(sizeof(tmp));
                      sprintf(string,"%d",tmp);
                      lista=aggiungi_elemento(lista,string);
                      op1=op1->punBack;
```

```
                }
            }
        }
        else{
            if(n_colonne==n_righe&&n_righe==1){
                if(tipo==1)
                    op2f=(float )op2i;
            }
            else{
                if(vcolonne==vrighe&&vrighe==1){
                    if(type==1)
                        op2f=(float )op2i;
                }
            }
            while(op1!=NULL){
                if(strcmp(op1->string,"/")==0)
                {
                    lista=aggiungi_elemento(lista,"/");
                    op1=op1->punBack;
                }
                else{
                    string=NULL;
                    string = (char*)malloc(sizeof(op1->string));
                    strcpy(string,op1->string);
                    float tmp=op2f-atof(string);
                    if(n_colonne==n_righe&&n_righe==1)
                        tmp=tmp*(-1);
                    string=NULL;
                    string = (char*)malloc(sizeof(tmp));
                    sprintf(string,"%f",tmp);
                    lista=aggiungi_elemento(lista,string);
                    op1=op1->punBack;
                }
            }
        }

        return lista;
}

struct elementoLista *differenza(struct elementoLista *op1, struct
elementoLista *op2,int tipo,int type,int n_colonne,int n_righe,int
vcolonne, int vrighe){

        struct elementoLista *lista = NULL;
        char *string=NULL;

        if(vcolonne==n_colonne&&vrighe==n_righe){
            lista=NULL;
            int tmpi=0;
            float tmpf=0;
            while(op1!=NULL&&op2!=NULL)
            {
                if(strcmp(op1->string,"/")==0&&strcmp(op2-
>string,"/")==0)
                {
                    lista=aggiungi_elemento(lista,"/");
                }
                else{
                    if(tipo==type&&tipo==1){
                        tmpi=atoi(op1->string)-atoi(op2->string);
                        string=NULL;
                        string=(char *)malloc(sizeof(tmpi));
                        sprintf(string,"%d",tmpi);
                    }
```

```
                        else{
                              tmpf=atof(op1->string)-atof(op2->string);
                              string=NULL;
                              string=(char *)malloc(sizeof(tmpf));
                              sprintf(string,"%f",tmpf);
                        }
                        lista=aggiungi_elemento(lista,string);
                  }
                  op1=op1->punBack;
                  op2=op2->punBack;
                  tmpi=0;
                  tmpf=0;
            }
            if(op1!=NULL||op2!=NULL){
                  printf("\nError: Unexpected error.\n");
                  exit(0);
            }
      }
      else{
            printf("\nError using mtimes: Inner matrix dimensions must
agree.\n");
            exit(0);
      }

      return lista;
}

struct elementoLista *somma_scalare_vettore(struct elementoLista *op1,
int op2i, float op2f,int tipo,int type,int n_colonne,int n_righe,int
vcolonne, int vrighe){

      struct elementoLista *lista = NULL;
      char *string=NULL;

      if(tipo==1&&type==1){
            while(op1!=NULL){
                  if(strcmp(op1->string,"/")==0)
                  {
                        lista=aggiungi_elemento(lista,"/");
                        op1=op1->punBack;
                  }
                  else{
                        string=NULL;
                        string = (char*)malloc(sizeof(op1->string));
                        strcpy(string,op1->string);
                        int tmp=op2i+atoi(string);
                        string=NULL;
                        string = (char*)malloc(sizeof(tmp));
                        sprintf(string,"%d",tmp);
                        lista=aggiungi_elemento(lista,string);
                        op1=op1->punBack;
                  }
            }
      }
      else{
            if(n_colonne==n_righe&&n_righe==1){
                  if(tipo==1)
                        op2f=(float )op2i;
            }
            else{
                  if(vcolonne==vrighe&&vrighe==1){
                        if(type==1)
                              op2f=(float )op2i;
                  }
```

```
            }
            while(op1!=NULL){
                  if(strcmp(op1->string,"/")==0)
                  {
                        lista=aggiungi_elemento(lista,"/");
                        op1=op1->punBack;
                  }
                  else{
                        string=NULL;
                        string = (char*)malloc(sizeof(op1->string));
                        strcpy(string,op1->string);
                        float tmp=op2f+atof(string);
                        string=NULL;
                        string = (char*)malloc(sizeof(tmp));
                        sprintf(string,"%f",tmp);
                        lista=aggiungi_elemento(lista,string);
                        op1=op1->punBack;
                  }
            }
      }

      return lista;
}

struct elementoLista *somma(struct elementoLista *op1, struct
elementoLista *op2,int tipo,int type,int n_colonne,int n_righe,int
vcolonne, int vrighe){

      struct elementoLista *lista = NULL;
      char *string=NULL;

      if(vcolonne==n_colonne&&vrighe==n_righe){
            lista=NULL;
            int tmpi=0;
            float tmpf=0;
            while(op1!=NULL&&op2!=NULL)
            {
                  if(strcmp(op1->string,"/")==0&&strcmp(op2-
>string,"/")==0)
                  {
                        lista=aggiungi_elemento(lista,"/");
                  }
                  else{
                        if(tipo==type&&tipo==1){
                              tmpi=atoi(op1->string)+atoi(op2->string);
                              string=NULL;
                              string=(char *)malloc(sizeof(tmpi));
                              sprintf(string,"%d",tmpi);
                        }
                        else{
                              tmpf=atof(op1->string)+atof(op2->string);
                              string=NULL;
                              string=(char *)malloc(sizeof(tmpf));
                              sprintf(string,"%f",tmpf);
                        }
                        lista=aggiungi_elemento(lista,string);
                  }
                  op1=op1->punBack;
                  op2=op2->punBack;
                  tmpi=0;
                  tmpf=0;
            }
            if(op1!=NULL||op2!=NULL){
                  printf("\nError: Unexpected error.\n");
```

```
                  exit(0);
            }
      }
      else{
            printf("\nError using mtimes: Inner matrix dimensions must
agree.\n");
            exit(0);
      }

      return lista;
}
```

Grammatica

```
/* Simbolo iniziale della grammatica */
input: /**/
      | input start
;

start: statement_list PUNTOEVIRGOLA NEWLINE
      |statement_list NEWLINE
      |NEWLINE
;

statement_list: statement_list PUNTOEVIRGOLA stm
      |statement_list VIRGOLA stm
      |stm
      /*Gestione errori*/
      |stm " " stm
      /*Fine gestione errori*/
;

stm: ID UGUALE assignment
      |vet_list
      |ID
      |SIZE TONDAAPERTA ID TONDACHIUSA
      |SIZE TONDAAPERTA vet_list TONDACHIUSA
      |CLEAR
      /*Gestione errori sintattici*/
      | vet_list vet_list
      | vet_list ID
      | ID vet_list
      | ID ID
      | vet_list errore
      | ID UGUALE
      |ID UGUALE APICE
      | errore
      |QUADRAAPERTA row_lists QUADRACHIUSA
      /*Fine gestione*/
;

assignment:QUADRAAPERTA row_lists QUADRACHIUSA
      |QUADRAAPERTA QUADRACHIUSA
      | op_arit
;

errore: UGUALE
      | UGUALE vet_list
      | UGUALE ID
;

row_lists: row_lists PUNTOEVIRGOLA vet_lists
      |row_lists NEWLINE vet_lists
      |row_lists PUNTOEVIRGOLA NEWLINE vet_lists
      | vet_lists
;

vet_lists: vet_lists VIRGOLA vet_list
      |vet_lists vet_list
      | vet_list
      /*Gestione errori*/
      |vet_lists ID
      |vet_lists VIRGOLA ID
      |ID
      /*Fine gestione errori*/
```

```
;

vet_list: type_num
      |STR_CHAR
;

type_num: PIU UNSIGNED_NUMBER
      |MENO UNSIGNED_NUMBER
      |MENO FLOAT_NUMBER
      |PIU FLOAT_NUMBER
      |UNSIGNED_NUMBER
      |FLOAT_NUMBER
;

op_arit : op_arit PIU term
      |op_arit MENO term
      |term
;

term: term PER operando
      |term DIVISO operando
      |operando
;

operando: vet_list
      | operando APICE
      |ID
;
```

CASI DI TEST

def_variabili.txt

```
% script di esempio per la definizione di variabili
% %

a=1;
b=+1
b=-2;
d='dddf-';
p=[]
e=[1 2 3]
s = [1 7, 3];
f=[-3 -5 +4];
u = [3,5
4,6];
c=3.4;
g=[4.3 4 2.2];
h=['dd']
f=['re', 'fg', 'wwe','hhh'];
i=[4,2,9.334;8 3 89];
l=['dfg';'qwe'];
n=b;
i=['dd' 'dfg']
l=['dd' 'dfg';'wwwww']
r=8; f=3;
j='ggg',m=6.7;
z = '|!"£$%&/()=?^\ì€éè[*+]çò@°à#§ù<>;:_,.-{}]';
6.7
-0.345;
'sdf';
s=i;
y=[' quel' ' ra' 'mo' ; ' del ' 'la' 'go ']
x=['ciaociao123'
'ciccio' '12_*_'];



a
b
c
d
e
f
g
h
i
j
l
m
n
p
r
s
u
x
y
z
ans

clear
```

op_aritmetiche_addizione.txt

```
% script di esempio per le operazioni aritmetiche (addizione)

a = [15 27;3 12];
b = [-2; -4; -5]
c = [0.4 ; -0.6]
d = ['loop';'golp'];
e='ciao'
j=[]
m=8
n=-7
o=0.33

a
b
c
d
e
j
m
n
o

t = 1+2;
u = 4+-4;
v=2+0.5;
x=7+t;
l=4+u;
y=2+v;
z=3+a;
g = 3+b;
h = 10+c;
k=5+j;

t
u
v
x
l
y
z
g
h
k

t = -1+2;
u = -4+-4;
v=-2+0.5;
x=-7+t;
l=-4+u;
y=-2+v;
z=-3+a;
g = -3+b;
h = -10+c;
k=-5+j;

t
u
v
x
l
y
```

```
z
g
h
k

t = -1.2+2;
u = -4.1+-4;
v=-2.1+0.5;
x=-7.3+t;
l=-4.2+u;
y=-2.1+v;
z=-3.3+a;
g = -3.5+b;
h = -10.8+c;
k=-5.6+j;

t
u
v
x
l
y
z
g
h
k

t = m+2;
u = m+-4;
v=m+0.5;
x=m+t;
l=m+u;
y=m+v;
z=m+a;
g = m+b;
h = m+c;
k=m+j;

t
u
v
x
l
y
z
g
h
k

t = n+2;
u = n+-4;
v=n+0.5;
x=n+t;
l=n+u;
y=n+v;
z=n+a;
g = n+b;
h = n+c;
k=n+j;

t
u
v
x
```

```
l
y
z
g
h
k

t = o+2;
u = o+-4;
v=o+0.5;
x=o+t;
l=o+u;
y=o+v;
z=o+a;
g = o+b;
h = o+c;
k=o+j;

t
u
v
x
l
y
z
g
h
k

T = a+2;
U = a+-4;
V=a+0.5;

T
U
V

T = b+2;
U = b+-4;
V=b+0.5;

T
U
V

T = c+2;
U = c+-4;
V=c+0.5;

T
U
V

T = j+2;
U = j+-4;
V=j+0.5;

T
U
V

x=a+m;
l=a+n;
y=a+o;
```

```
x
l
y

z=a+a;
-6
s = ans+a;

z
ans
s

x=j+m;
l=j+n;
y=j+o;

x
l
y

k=j+j;
-6
s = ans+j;

k
ans
s

%casi di errore
k=a+j;
z=j+a;
h = j+c;
```

op_aritmetiche_sottrazione.txt

```
% script di esempio per le operazioni aritmetiche (sottrazione)

a = [15 27;3 12];
b = [-2; -4; -5]
c = [0.4 ; -0.6]
d = ['loop';'golp'];
e='ciao'
j=[]
m=8
n=-7
o=0.33

a
b
c
d
e
j
m
n
o

t = 1-2;
u = 4--4;
v=2-0.5;
x=7-t;
l=4-u;
y=2-v;
z=3-a;
g = 3-b;
h = 10-c;
k=5-j;

t
u
v
x
l
y
z
g
h
k

t = -1-2;
u = -4--4;
v=-2-0.5;
x=-7-t;
l=-4-u;
y=-2-v;
z=-3-a;
g = -3-b;
h = -10-c;
k=-5-j;

t
u
v
x
l
y
```

```
z
g
h
k

t = -1.2-2;
u = -4.1--4;
v=-2.1-0.5;
x=-7.3-t;
l=-4.2-u;
y=-2.1-v;
z=-3.3-a;
g = -3.5-b;
h = -10.8-c;
k=-5.6-j;

t
u
v
x
l
y
z
g
h
k

t = m-2;
u = m--4;
v=m-0.5;
x=m-t;
l=m-u;
y=m-v;
z=m-a;
g = m-b;
h = m-c;
k=m-j;

t
u
v
x
l
y
z
g
h
k

t = n-2;
u = n--4;
v=n-0.5;
x=n-t;
l=n-u;
y=n-v;
z=n-a;
g = n-b;
h = n-c;
k=n-j;

t
u
v
x
```

```
l
y
z
g
h
k

t = o-2;
u = o--4;
v=o-0.5;
x=o-t;
l=o-u;
y=o-v;
z=o-a;
g = o-b;
h = o-c;
k=o-j;

t
u
v
x
l
y
z
g
h
k

T = a-2;
U = a--4;
V=a-0.5;

T
U
V

T = b-2;
U = b--4;
V=b-0.5;

T
U
V

T = c-2;
U = c--4;
V=c-0.5;

T
U
V

T = j-2;
U = j--4;
V=j-0.5;

T
U
V

x=a-m;
l=a-n;
y=a-o;
```

```
x
l
y

z=a-a;
-6
s = ans-a;

z
ans
s

x=j-m;
l=j-n;
y=j-o;

x
l
y

k=j-j;
-6
s = ans-j;

k
ans
s


%casi di errore
k=a-j;
z=j-a;
h = j-c;
```

op_aritmetiche_moltiplicazione.txt

```
% script di esempio per le operazioni aritmetiche (moltiplicazione)

a = [15 27;3 12];
b = [-2; -4; -5]
c = [0.4 ; -0.6]
d = ['loop';'golp'];
e='ciao'
j=[]
m=8
n=-7
o=0.33

a
b
c
d
e
j
m
n
o

t = 1*2;
u = 4*-4;
v=2*0.5;
x=7*t;
l=4*u;
y=2*v;
z=3*a;
g = 3*b;
h = 10*c;
k=5*j;

t
u
v
x
l
y
z
g
h
k

t = -1*2;
u = -4*-4;
v=-2*0.5;
x=-7*t;
l=-4*u;
y=-2*v;
z=-3*a;
g = -3*b;
h = -10*c;
k=-5*j;

t
u
v
x
l
y
```

```
z
g
h
k

t = -1.2*2;
u = -4.1*-4;
v=-2.1*0.5;
x=-7.3*t;
l=-4.2*u;
y=-2.1*v;
z=-3.3*a;
g = -3.5*b;
h = -10.8*c;
k=-5.6*j;

t
u
v
x
l
y
z
g
h
k

t = m*2;
u = m*-4;
v=m*0.5;
x=m*t;
l=m*u;
y=m*v;
z=m*a;
g = m*b;
h = m*c;
k=m*j;

t
u
v
x
l
y
z
g
h
k

t = n*2;
u = n*-4;
v=n*0.5;
x=n*t;
l=n*u;
y=n*v;
z=n*a;
g = n*b;
h = n*c;
k=n*j;

t
u
v
x
```

```
l
y
z
g
h
k

t = o*2;
u = o*-4;
v=o*0.5;
x=o*t;
l=o*u;
y=o*v;
z=o*a;
g = o*b;
h = o*c;
k=o*j;

t
u
v
x
l
y
z
g
h
k

T = a*2;
U = a*-4;
V=a*0.5;

T
U
V

T = b*2;
U = b*-4;
V=b*0.5;

T
U
V

T = c*2;
U = c*-4;
V=c*0.5;

T
U
V

T = j*2;
U = j*-4;
V=j*0.5;

T
U
V

x=a*m;
l=a*n;
y=a*o;
```

```
x
l
y

z=a*a;
h = a*c;
k=a*j;
-6
s = ans*a;

z
h
k
ans
s

x=j*m;
l=j*n;
y=j*o;

x
l
y

z=j*a;
h = j*c;
k=j*j;
-6
s = ans*j;

z
h
k
ans
s
```

op_aritmetiche_divisione.txt

```
% script di esempio per le operazioni aritmetiche (divisione)

a = [15 27;3 12];
b = [-2; -4; -5]
c = [0.4 ; -0.6]
d = ['loop';'golp'];
e='ciao'
j=[]
m=8
n=-7
o=0.33

a
b
c
d
e
j
m
n
o

t = 1/2;
u = 4/-4;
v=2/0.5;
x=7/t;
l=4/u;
y=2/v;

t
u
v
x
l
y

t = -1/2;
u = -4/-4;
v=-2/0.5;
x=-7/t;
l=-4/u;
y=-2/v;

t
u
v
x
l
y

t = -1.2/2;
u = -4.1/-4;
v=-2.1/0.5;
x=-7.3/t;
l=-4.2/u;
y=-2.1/v;

t
u
v
x
```

```
l
y

t = m/2;
u = m/-4;
v=m/0.5;
x=m/t;
l=m/u;
y=m/v;

t
u
v
x
l
y

t = n/2;
u = n/-4;
v=n/0.5;
x=n/t;
l=n/u;
y=n/v;

t
u
v
x
l
y

t = o/2;
u = o/-4;
v=o/0.5;
x=o/t;
l=o/u;
y=o/v;

t
u
v
x
l
y

T = a/2;
U = a/-4;
V=a/0.5;

T
U
V

T = b/2;
U = b/-4;
V=b/0.5;

T
U
V

T = c/2;
U = c/-4;
V=c/0.5;
```

```
T
U
V

T = j/2;
U = j/-4;
V=j/0.5;

T
U
V

x=a/m;
l=a/n;
y=a/o;

x
l
y

x=j/m;
l=j/n;
y=j/o;

x
l
y
```

trasposta e size.txt

```
% script di esempio per le operazioni di trasposizione e utilizzo della
funzione size

a = [1 2 3];
a
b = a';
b
c = [];
d = c';
d
e = 6.7';
e
F=['bye' 'bye'];
F
G=F';
G
h = 'bbb';
h=h';
h

size(a)
size('pippo')
```

prova.txt

```
a=2;
b=3;
c=4;

t=a*b+c;
t

s = a+b*c;
s

h=a*b+8/c+5;
h

k=1+a+b;
k

k=a*a*b;
k

k=a/a*b;
k

k=10/a-c;
k

k=10/a-c/a;
k

t=2*3+4;
t

s = 2+3*4;
s

h=2*3+8/4+5;
h

k=1+2+3;
k

k=2*2*3;
k

k=2/2*3;
k

k=10/2-4;
k

k=10/2-4/2;
k

j=[];
L=j;
L

a=[2 3;2 5];
R = c*a';
R

U=a'*c;
U
```

```
T=a*a*a;
T

P=a*a+a;
P

Q=a*a+a';
Q

V=a*a+a'-a;
V

V=a*a+a'*a;
V

s=['dd' 'gg';'y' 'rtt'];
s

E=s';
E

E=[1 2];
F=[3 4];
G=[4 6];

A= E+F+G;
A

d=8/4/2;
d

A=2';
A
A=-3';
A
A=-0.8';
A

B=[1 2 3;4 5 6];
A=B';
A

B=[1 2];
A=B';
A

B=[1;2];
A=B';
A

B=[];
B
A=B';
A

B= 7;
B
A=B';
A

s=2+4*6;
s
```

errore.txt

Di seguito riportiamo i tipi di errori riscontrabili e alcuni esempi relativi a ciascun tipo. Come file per i casi di test verranno allegati solo alcuni di essi, vista la vastità di combinazioni di casi di errore che è possibile avere. Tutti quelli di seguito riportati sono stati comunque testati con esito positivo.

```
%Error: Undefined function or variable

f
b = f

%Error: Unexpected MATLAB expression

4 8

-8 h

c = 4 8
c = 9 h

%Error: Undefined function or method 'k' for input arguments of type
'char'

g 'jhf'

Iuu v

%Error: The expression to the left of the equals sign is not a valid
target for an assignment

56 =
56 = 8
56 = t
=
= 8
= t

%Error: Expression or statement is incomplete or incorrect

d =

%Error: A MATLAB string constant is not terminated properly

j = '

%Error: The expression is not allowed.

[7 5 3]
s = [4 2 a]
s = [4, 3, v]
g = 'dfg'*3
j = 'hhhh'
k=2*j
g = 'dfg'/3
j = 'hhhh'
k=2/j
e = 67/0
x = 0
f = 89/x
D = [2 5 4]
```

```
z = 77 / D;
g = 'dfg'+3
j = 'hhhh'
k=2+j
g = 'dfg'-3
j = 'hhhh'
k=2-j

%Error using mtimes: Inner matrix dimensions must agree

A = [3 4]
B = [3;4;5]
C = A*B
C = A+B
C = A-B

%Error using vertcat: CAT arguments dimensions are not consistent

A = [1,5,3;6]
B = ['ddd' 'gg';'ba']
```

BIBLIOGRAFIA

TOM NIEMANN, *A Compact Guide To Lex & Yacc*, versione PDF disponibile presso ePaperPress.com

© COPYRIGHT 1984–2008 by THE MATHWORKS, INC., *Documentation for MathWorks Products R2008b*, http://www.mathworks.com/access/helpdesk/help/helpdesk.html

GIACOMO PISCITELLI, *Dispense del corso di "Compilatori ed Interpreti"*, A.A. 2009-10