# Compilatori ed interpreti

AA 2009/2010


Docente: Prof. Giacomo Piscitelli

progetto a cura di:

V. Renò - A. Troccoli

## *Sommario*

## *Introduzione*

L'Objective C è un linguaggio di programmazione orientato agli oggetti, dinamico, non fortemente tipizzato, sviluppato da Brad Cox alla metà degli anni ’80 presso la Stepstone Corporation, che deriva da SmallTalk, il celeberrimo capostipite di molti linguaggi Object Oriented, e da C, con il quale mantiene una completa compatibilità.

È dinamico perché posticipa a runtime la maggior parte delle azioni sugli oggetti. La conoscenza di come trattare a runtime gli oggetti deriva dal Runtime: una libreria che viene staticamente linkata ad ogni programma ObjectiveC. In sostanza il Runtime agisce come una specie macchina virtuale in cui "vivono" gli oggetti ObjectiveC.

Il Runtime mette a disposizione delle API che consentono ad esempio di conoscere per ogni oggetto a quale classe appartiene, i metodi e le proprietà che possiede ect.; queste caratteristiche avvicinano più Objective C a linguaggi tipo il Ruby o Python piuttosto che C++.

Essendo, quindi, un superset di C in Objective C si possono tranquillamente utilizzare tutti i metodi e le funzioni del C in maniera nativa. Ne consegue che è possibile compilare un qualsiasi programma scritto in C con un compilatore Objective C.

La gran parte della sintassi (clausole del preprocessore, espressioni, dichiarazioni e chiamate di funzioni) è derivata da quella del C, mentre la sintassi relativa alle caratteristiche object-oriented è stata creata per ottenere la comunicazione a scambio di messaggi simile a quella di Smalltalk.

Il modello di programmazione dell'Objective C è basato sullo scambio di messaggi e per questo si differenzia da quello di Simula, che viene usato in numerosi linguaggi quali, tra gli altri, il C++. Questa distinzione è semanticamente importante e consiste principalmente nel fatto che in Objective C non si *chiama un metodo*, ma si *invia un messaggio*.

Si dice che un oggetto chiamato *ogg* la cui classe implementa il metodo *faiQualcosa*, *risponde* al messaggio *faiQualcosa*. L'invio del messaggio *faiQualcosa* all'oggetto *ogg* è espresso da:

[ogg faiQualcosa];

mentre l'azione equivalente in C++ sarebbe espressa da:

ogg.faiQualcosa();

In questo modo è possibile inviare messaggi ad un oggetto anche se l'oggetto *non è capace* di rispondere. Questo differisce dai linguaggi tipizzati staticamente come C++ e Java nei quali tutte le chiamate devono essere di metodi predefiniti.

L'Objective C richiede che l'interfaccia e l'implementazione di una classe siano dichiarati in blocchi di codice differenti. Per convenzione l'interfaccia è messa in un file con suffisso ".h", mentre l'implementazione in un file con suffisso ".m". Si è soliti assegnare il nome ai file basandosi sul nome della classe, nell'esempio "NomeDellaClasse.h".

```
//definizione dell'interfaccia: "NomeDellaClasse.h"
#import "NomeDellaSuperclasse.h"

@interface NomeDellaClasse : NomeDellaSuperclasse
{
   //variabili d'istanza
   int variabileIntera;
   float variabileFloat;
   ...
}
//metodi di classe
+ metodoDiClasse1
+ metodoDiClasse2
+ ...

//metodi di istanza
- metodoDiIstanza1
- metodoDiIstanza2
- ...

@end
```

Il segno meno (-) denota i metodi d'istanza, mentre il segno più (+) quelli di classe (analoghi alle funzioni statiche del C++). Si noti la differenza di significato con le convenzioni dei diagrammi UML dove i due segni rappresentano rispettivamente i metodi privati e pubblici.

L'interfaccia dichiara solamente i prototipi dei metodi e non i metodi stessi che vengono inseriti nell'implementazione, scritta in un file con estensione ".m". La convenzione usata è quella di assegnare il nome al file basandosi sul nome della classe, nell'esempio "NomeDellaClasse.m"

```
//definizione dell'implementazione: "NomeDellaClasse.m"
#import "NomeDellaClasse.h"

@implementation NomeDellaClasse
+ metodoDiClasse1
{
   // implementazione
   ...
}
 + metodoDiClasse2
{
   // implementazione
   ...
}
...
- metodoDiIstanza1
{
   // implementazione
   ...
}
- metodoDiIstanza2
{
   // implementazione
   ...
```

```
}
...
@end
```

I metodi sono scritti in maniera diversa dalle funzioni in stile C. Ad esempio, una funzione, sia in C che in Objective C segue la seguente forma generale:

```
int fai_la_radice_quadrata(int i)
{
   return radice_quadrata(i);
}
```

che avrà come prototipo:

```
int fai_la_radice_quadrata(int);
```

L'implementazione come metodo diverrà:

```
- (int) fai_la_radice_quadrata:(int) i
{
   return [self radice_quadrata: i];
}
```

Un approccio più canonico alla scrittura del metodo sarebbe quello di citare il primo argomento nel nome del selettore:

```
- (int) faiLaRadiceQuadrataDiInt: (int) i
{
   return [self radiceQuadrataDiInt:i];
}
```

Questa sintassi può apparire complicata, ma consente di assegnare i nomi ai parametri, ad esempio:

```
- (int) changeColorWithRed:(int) r  green:(int) g  blue:(int) b
```

può essere invocato così:

```
[myColor changeColorWithRed:5 green:2 blue:6];
```

Le rappresentazioni interne di questi metodi possono variare con le diverse implementazioni di Objective C.

L'implementazione dell'Objective C usa un semplice run-time system scritto in linguaggio C che aumenta di poco la dimensione delle applicazioni. Al contrario, la maggior parte dei sistemi object-oriented esistenti quando fu creato (e Java tuttora) usava una grossa macchina virtuale invasiva per l'intero sistema. I programmi scritti in Objective C tendono a non essere troppo più grandi delle dimensioni del loro codice oggetto e delle librerie usate (che generalmente non devono essere incluse nel codice distribuito), al contrario ad esempio dei sistemi Smalltalk dove grandi quantità di memoria sono necessarie semplicemente per aprire una finestra.

Il linguaggio può essere implementato con un compilatore C (in GCC, prima come un preprocessore ed in seguito come un modulo del compilatore) piuttosto che con un nuovo compilatore. Ciò consente all'Objective C di sfruttare l'enorme mole di codice, librerie e strumenti già esistenti in C che può essere adattata in Objective C per fornire un'interfaccia object-oriented. Tutte questi fattori riducono le barriere d'ingresso al nuovo linguaggio, fattore che costituì il problema principale di Smalltalk negli anni '80.

Le prime versioni di Objective C non supportavano la garbage collection. Al tempo questa scelta fu oggetto di discussioni e in molti (ai tempi di Smalltalk) la consideravano un lungo "tempo morto" in cui il sistema era reso inusabile. Anche se qualche implementazione di terze parti (principalmente GNUstep) aveva già aggiunto questa caratteristica, questa è stata implementata da Apple in Mac OS X 10.5, ma non è disponibile per applicazioni implementate per versioni precedenti del sistema operativo.

Un'altra critica comunemente fatta all'Objective C è quella di non avere un supporto nativo per i namespace. I programmatori sono perciò costretti ad aggiungere prefissi in maniera più o meno arbitraria ai nomi delle classi che implementano, fatto che può

causare collisioni. Dal 2007 tutte le classi e le funzioni di Mac OS X in ambiente Cocoa hanno il prefisso "NS" (es. *NSObject* o *NSButton*) per identificarle chiaramente; "NS" deriva dal nome delle classi definite durante lo sviluppo di NeXTSTEP.

## *Implementazione*

Per l'implementazione di blocK si è fatto uso di due tool di supporto per la generazione di uno scanner e di un parser: Flex e Bison.

Il primo si occupa della parte relativa all'analizzatore lessicale (scanner) mediante il riconoscimento di pattern all'interno dello stream di input.

Flex lavora riconoscendo una serie di token descritti per mezzo di espressioni regolari (rules) eventualmente corredate di codice C da eseguire in corrispondenza del pattern riconosciuto.

L'output di Flex è un sorgente C chiamato di default "lex.yy.c", il quale definisce la routine yylex(). L'input di Flex (file .l) si compone di tre sezioni separate da una linea contenente il simbolo "%%".

1. Definizione: sezione contenente le dichiarazioni facoltative di nomi che identifichino pattern predefiniti (ad esempio un digit può essere identificato dall'epressione regolare [0-9]).
2. Regole: sezione contenente una serie di regole condizione-azione, la condizione è identificata mediante espressione regolare, l'azione è implementata in questo caso con un
   return NOME_TOKEN;

   dove NOME_TOKEN è una macro definita automaticamente dai tools.
3. Codice utente: è la sezione conclusiva dove il programmatore ha la possibilità di implementare logiche differenti da quelle di default per quel che riguarda le azioni compiute dallo scanner.

Anziché produrre il semplice eseguibile dello scanner, la routine yylex() viene richiamata a runtime dall'analizzatore sintattico di blocK, generato mediante l'ausilio del tool Bison, versione open source del più celebre Yacc (Yet Another Compiler of Compilers); il lettore più attento noterà sicuramente il simpatico gioco di parole fra i due nomi degli omologhi tools.

Bison si occupa di convertire una grammatica libera da contesto scritta in BNF nel corrispettivo parser producendo una funzione C che svolga questo compito.

Un file di input del tool ha estensione ".y" e si compone, come accade per Flex, di più parti:

1. Prologo: sezione preliminare che contiene la definizione delle macro, la dichiarazione di funzioni e variabili che saranno richiamate in seguito (in maniera analoga a quanto accade per un programma C).
2. Dichiarazioni di Bison: sezione che contiene le dichiarazioni che definiscono terminali e non terminali associati alla grammatica e che, nell'uso combinato con flex, devono contenere un'istruzione del tipo:

   %token NOME_TOKEN

   per ogni valore di ritorno dello scanner.
3. Regole grammaticali: sezione che definisce le regole di derivazione tramite le quali il parser è in grado di generare l'albero sintattico. Una regola è espressa nella forma:

   NOME_REGOLA

   : espr1 {codice1}

   | espr2 {codice2}

   (…)

   | esprN {codiceN}

   ;
4. Epilogo: sezione che contiene ulteriore codice C che verrà copiato in coda al parser generato e che, generalmente, è utilizzata per ridefinire le funzioni che altrimenti sarebbero limitate a quelle di default.

Ogni sezione è delimitata da "%%".

Nel momento in cui viene chiamata la funzione yyparse(), quest'ultima provvede alla chiamata di yylex() affinchè sia possibile iniziare la lettura dell'input segmentato in token.

Bison inserisce in una pila i token ricevuti (operazione di shift) e, nel frattempo, cerca di raggrupparli per ridurli (operazione di reduce) secondo una delle regole definite dalla grammatica.

Partendo dalle foglie dell'albero quindi, si cerca di ridurre progressivamente verso l'assioma seguendo ovviamente un approccio bottom up.

Bison inoltre prevede un lookahead di un token, pertanto, anziché ridurre con la prima regola possibile, cerca di esaminare il token successivo per decidere la migliore strategia di derivazione da effettuare. Nel caso in cui un token di lookahead non sia sufficiente a discriminare in maniera opportuna una o più alternative dell'albero sintattico, Bison genera un warning chiamato "shift/reduce" che non implica necessariamente la presenza di ambiguità nella grammatica in quanto può essere generato anche da semplici ricorsioni che si rivelano necessarie durante la scrittura della stessa.

Un errore assolutamente da evitare, invece, è il conflitto "reduce/reduce" che indica una situazione per la quale vi sono effettivamente due o più strade possibili da seguire nell'albero per ridurre una stessa regola e che quindi identificano una situazione di forte ambiguità.

## *Lo script di compilazione*

Una volta installati i tools, si è reso necessario creare il file "compile.sh" che contiene le seguenti direttive:

- bison -k -g -d parser.y
  crea il file parser.tab.c a partire da parser.y con i seguenti parametri:
  - –k: incude una tabella dei nomi dei token
  - –g: produce una descrizione dell'automa in formato VCG, visualizzabile ad esempio tramite il tool aiSee
  - –d: produce un file header, indispensabile per il funzionamento combinato con l'analizzatore lessicale
- gcc -c parser.tab.c
  genera il solo codice oggetto corrispondente a parser.tab.c
- flex --yylineno scanner.l
  crea il file lex.yy.c a partire da scanner.l
  - -- yylineno: tiene traccia del numero di linee che compongono il file di input
- gcc -c lex.yy.c
  genera il solo codice oggetto corrispondente a lex.yy.c
- gcc parser.tab.o lex.yy.o -o blocK
  genera l'eseguibile blocK a partire dai due codici oggetto

Un ulteriore script chiamato "compiled.sh" si occupa di creare lo stesso eseguibile in modalità debug. Questo programma deve essere eseguito esclusivamente da riga di comando e consete di esplorare l'albero sintattico passo passo mantenendo inoltre traccia dello stato dell'automa. Esso differisce dal precendente unicamente per il flag --debug inserito tra le opzioni del comando bison.

## *Scanner.l*

### La sezione *blocco di codice*

```
%{
#include <stdio.h>
#include <stdlib.h>
#include "parser.tab.h"


void count();
void comment();
void commento();
int check_type();
char *p;
int campo;
int conta;
extern int numeroriga;
extern struct token_stack *ts_head;
extern char *r;
char *c;


%}
```

Oltre alle librerie standard si nota la presenza dell'header del parser, infatti tramite questa inclusione è possibile far funzionare in maniera combinata i due tools. Successivamente sono presenti i prototipi delle funzioni che saranno implementate nella terza sezione e le dichiarazioni delle variabili che saranno utilizzate in seguito; alcune di queste (le extern) fanno riferimento alle omonime presenti nel parser.

### La sezione *definizione*

```
DIGIT           [0-9]
LETTER          [a-zA-Z_]
HEX             [a-fA-F0-9]
EXP             [Ee][+-]?{DIGIT}+
FLSIGN          (F|f|L|l)
INTSIGN         (u|U|l|L)*
```

Riscrive in maniera ottimizzata espressioni regolari ricorrenti come ad esempio numeri e lettere.

## La sezione delle *regole*

```
"/*"                    { comment(); }
"//"                   { commento();}
"auto"                  { count(); return(AUTO); }
"break"                 { count(); return(BREAK); }
"case"                  { count(); return(CASE); }
"char"                  { count(); push_token("char",yytext,CHAR);
return(CHAR); }
"const"                 { count(); push_token("const",yytext,CONST);
return(CONST); }
"continue"              { count(); return(CONTINUE); }
"default"               { count(); return(DEFAULT); }
"do"                    { count(); return(DO); }
"double"                { count(); push_token("double",yytext,DOUBLE);
return(DOUBLE); }
"else"                  { count(); return(ELSE); }
"enum"                  { count(); return(ENUM); }
"extern"                { count(); return(EXTERN); }
"float"                 { count(); push_token("float",yytext,FLOAT);
return(FLOAT); }
"for"                   { count(); return(FOR); }
"goto"                  { count(); return(GOTO); }
"if"                    { count(); return(IF); }
"int"                   { count();
push_token("int",yytext,INT);return(INT); }
"long"                  { count(); push_token("long",yytext,LONG);
return(LONG); }
"register"              { count(); return(REGISTER); }
"return"                { count(); return(RETURN); }
"short"                 { count(); push_token("short",yytext,SHORT);
return(SHORT); }
"signed"                { count(); return(SIGNED); }
"sizeof"                { count(); return(SIZEOF); }
"static"                { count(); return(STATIC); }
"struct"                { count(); return(STRUCT); }
"switch"                { count(); return(SWITCH); }
"typedef"               { count(); return(TYPEDEF); }
"union"                 { count(); return(UNION); }
"unsigned"              { count(); return(UNSIGNED); }
"void"                  { count();
push_token("void",yytext,VOID);return(VOID); }
"volatile"              { count(); return(VOLATILE); }
"while"                 { count(); return(WHILE); }

"#import"                  { count(); commento();}
"#include"              { count(); commento();}
"@class"                   {count(); return(OBJC_ATCLASS);}
"@interface"               {count(); return(OBJC_INTERFACE);}
"@property"                     {count(); return(OBJC_PROP);}
"@end"                          {count(); return(END);}
"@implementation"          {count(); return(OBJC_IMPLEMENTATION);}
"@synthesize"              {count(); return(OBJC_SYNTH);}

"nonatomic"                     {count(); return(OBJC_NONATOMIC);}
```

```
"retain"                    {count(); return(OBJC_RETAIN);}
"copy"                           {count(); return(OBJC_COPY);}
"assign"                  {count(); return(OBJC_ASSIGN);}
"readwrite"               {count(); return(OBJC_READWRITE);}
"readonly"                {count(); return(OBJC_READONLY);}
"super"                          {count(); return(OBJC_SUPER);}
"release"                   {count(); return(OBJC_RELEASE);}
"alloc"                          {count(); return(OBJC_ALLOC);}
"dealloc"                   {count(); return(OBJC_DEALLOC);}
"self"                           {count(); return(OBJC_SELF);}
"this"                           {count(); return(OBJC_THIS);}
"BOOL"                           {count();
push_token("objc_bool",yytext,OBJC_BOOL); return(OBJC_BOOL);}
"YES"                       {count(); return(OBJC_YES);}
"NO"                  {count(); return(OBJC_NO);}
"NULL"                           {count(); return(OBJC_NULL);}
"nil"                       {count(); return(OBJC_NIL);}
"init"                           {count(); return(OBJC_INIT);}

{LETTER}({LETTER}|{DIGIT})*            { count();    p=(char
*)calloc(strlen(yytext)+1,sizeof(char));
                                        strcpy(p,yytext);


                                       conta=controlla(p);

if(conta==0)
    {
        push_token("identifier",yytext,IDENTIFIER);
        return(IDENTIFIER);
    }
if(conta==1)
{

    campo=objc_tok(yytext);


    if(campo==1)
        return(OBJC_CLASS_METHOD);
    else if(campo==2)
    {
        push_token("objc_constant",yytext,CONSTANT);
        return(OBJC_CONSTANT);
    }
    else if(campo==3)
    {
        push_token("objc_data_type",yytext,OBJC_DATA_TYPE);
        return(OBJC_DATA_TYPE);
    }
    else if(campo==4)
        return(OBJC_DELEGATE_METHOD);
    else if(campo==5)
        return(OBJC_FUNCTION);
    else if(campo==6)
        return(OBJC_INSTANCE_PROPERTY);
    else if(campo==7)
        return(OBJC_PROTOCOL_METHOD);
```

```
    else if(campo==8)
        return(OBJC_VARI);
    else if(campo==9)
    {
        r=(char *)malloc(strlen(yytext)+1);
        strcpy(r,yytext);
        push_token("objc_class_name",yytext,OBJC_CLASS_NAME);
        return(OBJC_CLASS_NAME);
    }
    else if(campo==10)
        return(OBJC_INSTANCE_METHOD);
    else if(campo==11)
        return(IDENTIFIER);

                                        }
}
0[xX]{HEX}+{INTSIGN}?           { count(); return(HEX); }
0{DIGIT}+{INTSIGN}?             { count(); return(CONSTANT); }
{DIGIT}+{INTSIGN}?              { count(); return(CONSTANT); }
{LETTER}?'(\\.|[^\\'])+'        { count(); return(CONSTANT); }
{DIGIT}+{EXP}{FLSIGN}?          { count(); return(CONSTANT); }
{DIGIT}*"."{DIGIT}+({EXP})?{FLSIGN}?  { count(); return(CONSTANT); }
{DIGIT}+"."{DIGIT}*({EXP})?{FLSIGN}?  { count(); return(CONSTANT); }
{LETTER}?\"(\\.|[^\\"])*\"      { count(); return(STRING_LITERAL); }

"..."                   { count(); return(ELLIPSIS); }
">>="                   { count(); return(RIGHT_ASSIGN); }
"<<="                   { count(); return(LEFT_ASSIGN); }
"+="                    { count(); return(ADD_ASSIGN); }
"-="                    { count(); return(SUB_ASSIGN); }
"*="                    { count(); return(MUL_ASSIGN); }
"/="                    { count(); return(DIV_ASSIGN); }
"%="                    { count(); return(MOD_ASSIGN); }
"&="                    { count(); return(AND_ASSIGN); }
"^="                    { count(); return(XOR_ASSIGN); }
"|="                    { count(); return(OR_ASSIGN); }
">>"                    { count(); return(RIGHT_OP); }
"<<"                    { count(); return(LEFT_OP); }
"++"                    { count(); return(INC_OP); }
"--"                    { count(); return(DEC_OP); }
"->"                    { count(); return(PTR_OP); }
"&&"                    { count(); return(AND_OP); }
"||"                    { count(); return(OR_OP); }
"<="                    { count(); return(LE_OP); }
">="                    { count(); return(GE_OP); }
"=="                    { count(); return(EQ_OP); }
"!="                    { count(); return(NE_OP); }
";"                     { count(); return(';'); }
("{"|"<%")              { count(); return('{'); }
("}"|"%>")              { count(); return('}'); }
","                     { count(); return(','); }
":"                     { count(); return(':'); }
"="                     { count(); return('='); }
"("                     { count(); return('('); }
")"                     { count(); return(')'); }
("["|"<:")              { count(); return('['); }
("]"|":>")              { count(); return(']'); }
```

```
"."                     { count(); return('.'); }
"&"                     { count(); return('&'); }
"!"                     { count(); return('!'); }
"~"                     { count(); return('~'); }
"-"                     { count(); return('-'); }
"+"                     { count(); return('+'); }
"*"                     { count(); return('*'); }
"/"                     { count(); return('/'); }
"%"                     { count(); return('%'); }
"<"                     { count(); return('<'); }
">"                     { count(); return('>'); }
"^"                     { count(); return('^'); }
"|"                     { count(); return('|'); }
"?"                     { count(); return('?'); }

[ \t\v\f]             { count(); }
"\n"                   { count(); numeroriga++;}
.                       { /* ignore bad characters */ }

%%
```

Oltre i return che consentono l'invio dell'intero corrispondente al token verso il parser, sono presenti delle istruzioni che permettono:

- salvataggio delle informazioni relative al token in una pila aggiuntiva implementata nel parser (push_token(args))
- salvataggio della stringa corrispondente al token in un vettore p allocato dinamicamente che verrà richiamato nel parser per il controllo semantico
- implementazione di una logica che consenta di discriminare fra i vari costrutti dell'objective c effettuando query sulla Symbol Table.

Per questioni di semplicità, si è scelto di caricare la tabella dei simboli con un sottoinsieme di parole chiave afferenti ai framework di Cocoa più comunemente utilizzati. La discriminazione dei token propri dell'Objective C è quindi limitata ad UIKit e Foundation Framework.

La sezione *codice utente*

```
int column = 0;

void comment()
{
     char c0, c1;
    int flag=1;

        while(flag)
```

```
        {
            while ((c0= input()) != '*' && c0 != 0)
            {
                if(c0 == '\n')
                    numeroriga++;
                putchar(c0);
            }
            if ((c1 = input()) != '/' && c0 != 0)
            {
                unput(c1);
            }
            if(c1 == '/')
                flag=0;
        }

        if (c0 != 0)
            putchar(c1);
}


void count()
{
        int i;

        for (i = 0; yytext[i] != '\0'; i++)
                if (yytext[i] == '\n')
                        column = 0;
                else if (yytext[i] == '\t')
                        column += 8 - (column % 8);
                else
                        column++;
        ECHO;
}


void commento()
{
        char c0;
    int flag=1;

        while(flag)
        {
            while ((c0= input()) != '\n' && c0 != 0)
            {
                putchar(c0);
            }

            if(c0 == '\n'){
                numeroriga++;
                flag=0;
            }
        }

        if (c0 != 0)
            putchar(c0);
}
```

```
int yywrap() {  }
```

In questa sezione vengono definite le funzioni:

- comment() utile per rilevare commenti multi linea nel codice
- commento() identifica commenti su linea singola
- count() consente di contare il numero delle righe che compongono il file di input
- yywrap() funzione che viene chiamata quando lo stream di input raggiunge l'EOF

## *Regexutils.h*

Al fine di poter ricercare in maniera più performante una stringa che corrisponda ad un pattern, è essenziale importare la libreria regex.h che fornisce strutture dati in grado di manipolare espressioni regolari.

In "regexutils.h", inclusa nel progetto, viene in più definito il metodo match che permette di effettuare l'equivalente di una grep su shell bash all'interno di una stringa passata come input. Disporre di una funzione del genere ottimizza le ricerche nella tabella dei simboli consentendo una maggiore flessibilità rispetto alla classica funzione strcmp() disponibile nella libreria standard <string.h>.

```
#include <regex.h>

int match(const char *string, char *pattern)
{
    int    status;
    regex_t    re;

    if (regcomp(&re, pattern, REG_EXTENDED|REG_NOSUB) != 0) {
        return(0);      /* report error */
    }
    status = regexec(&re, string, (size_t) 0, NULL, 0);
    regfree(&re);
    if (status != 0) {
        return(0);      /* report error */
    }
    return(1);
}
```

## *STable.h*

La tabella dei simboli è una struttura dati usata da un compilatore allo scopo di tener traccia della semantica delle variabili. Questa può essere implementata in svariati modi, tra cui la lista non ordinata, la pila o anche l'albero binario.

Nonostante Bison dia la possibilità di generarne una a runtime in modo trasparente rispetto al programmatore, si è scelto comunque di implementarne una manualmente per poter gestire gli analizzatori più a basso livello.

Trattandosi di un linguaggio estremamante vasto e complesso, dato che non si è implementato il caricamento della tabella a partire dagli import o dagli include, quest'ultima viene precaricata con i tipi di dato relativi ai framework UIKit e Foundation, reperiti direttamente dal sito developer.apple.com

La tabella dei simboli di blocK, quindi, si configura come uno strumento in grado di fornire informazioni non solo riguardanti le variabili, ma che comprendano dati, metodi e protocolli tipici dei due framework o definiti dall'utente.

```
enum bool {false, true};

struct record
{
  char *nome;
  char *info;
  struct record *prox;
};
typedef struct record record;
record *stable = NULL;
void putsimbolo (char *simbolo, char *informazioni)
{
  record *punt;
  punt = (record *) malloc (sizeof(record));
  punt->nome = (char *) malloc (strlen(simbolo)+1);
  punt->info = (char *) malloc (strlen(informazioni)+1);
  strcpy (punt->nome, simbolo);
  strcpy (punt->info, informazioni);

  punt->prox = (struct record *)stable;
  stable = punt;
  }
```

```
record * getsimbolo (char *simbolo)
{
  record *punt;
  for ( punt = stable; punt!=NULL;punt = punt->prox )
  {
    if (strcmp (punt->nome,simbolo) == 0)
    {
        return punt;
    }

  }
  return NULL;
}


char * info_from_simbolo (char *simbolo)
{
    record *punt;
    char * inform=NULL;
    char buffer[256];
    int x=0;
    int d;


    buffer[0]='\0';
    strcpy(buffer,"^.*");

    strcat(buffer,simbolo);

    strcat(buffer,".*$");


  for ( punt = stable; punt!=NULL;punt = punt->prox )
  {
    if (strcmp (punt->nome,simbolo) == 0)
    {
        inform=punt->info;
     }
   }
   return inform;
}


int objc_tok(char *i)

{
    char *nom;
    int x=0;


    nom=info_from_simbolo(i);

        if(match(nom,"^.*class.*method.*$")==1)
            x= 1;
        else if(match(nom,"^.*constant.*$")==1)
            x= 2;
```

```
        else if(match(nom,"^.*data.*type.*$")==1)
            x= 3;
        else if(match(nom,"^.*delegate.*method.*$")==1)
            x= 4;
        else if(match(nom,"^.*function.*$")==1)
            x= 5;
        else if(match(nom,"^.*instance.*property.*$")==1)
            x= 6;
        else if(match(nom,"^.*protocol.*method.*$")==1)
            x= 7;
        else if(match(nom,"^.*vari.*$")==1)
            x= 8;
        else if(match(nom,"^.*class.*name.*$")==1)
            x= 9;
        else if(match(nom,"^.*instance.*method.*$")==1)
            x= 10;
        else
            x=11;

  return x;
}


void stampaelementi()
{
        record *p=NULL;
        int i=0;
        for(p=stable;p!=NULL;p=p->prox)
        {
             printf("nomesimbolo: %s infosssss: %s\n",p->nome,p->info);
             i++;
        }


}

void controlla_aggiungi( char *nuovo, char * inform)
{
    record * p=NULL;
    if(nuovo!=NULL)
    {
        p=getsimbolo(nuovo);
        if(p==0)
        {
            putsimbolo(nuovo,inform);

        }
    }else
    {
        printf("impossibile inserire nuovo simbolo\n");
        }


   }
```

```
int controlla( char *nuovo)
{
    record * p=NULL;
    int numero=0;
    if(nuovo!=NULL)
    {
        p=getsimbolo(nuovo);

        if(p==NULL)
        {
            numero= 0;

        }else
        {
            numero= 1;
            }
    }else
    {
        printf("impossibile inserire nuovo simbolo\n");
        numero= 2;
        }

     return numero;

}
```

Il file contiene la struttura del record che consta di un nome e di un tipo, il primo contenente l'identificatore relativo alla variabile, al metodo o al tipo di dato objc, il secondo invece contenente le informazioni a corredo; una variabile conterrà il suo tipo, un metodo il suo tipo di ritorno ed ogni token dell'objc una sua specificazione.

All'interno del parser si fa largo uso delle funzioni interne alla libreria "Stable.h" in quanto esse consentono l'accesso alla tabella in lettura e scrittura ed implementano ricerche al suo interno. La funzione "objc_tok" è quella che viene richiamata nello scanner per discriminare fra i vari identifier che si presentano nello stream di input: se questi trovano riscontro positivamente con uno dei pattern specificati, allora viene restituito un codice intero che permette di identificare il tipo di dato objc.

## *Parser.y*

La sezione *prologo:*

```
%{
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "regexutils.h"
#include "STable.h"


    extern char* yytext;
    extern char* p;
    extern int yylineno;

    struct token_stack
    {
     char *toke;
     char *tname;
     int ident;
     struct token_stack *prox;
    };
    struct token_stack *ts_head=NULL;
    struct record *rec=NULL;

    int numeroriga=1;
    int errori=0;
    int tipo_errore=0;

    char provv[256];
    char sp_tipo[128];
    char *vett_sx,*info_sx;
    char *vett_dx,*info_dx;
    char *p1,*q,*q1,*r,*a,*appino;

    int contatore;
    int conta_dichiaratore=0;
    int chiama_funz_errore=1;
    int returnflag=0;
    int flag1=0;
    int dealloc_flag=0;
    int constant_flag=0;
    int boolean_flag=0;
    int null_flag=0;
    int flag_funzione_puntatore=0;
    int flag_alloc=0;
    int flag_nome_funz=0;
    int num_argomenti=1;
    char n[128];

    int errore_tipo_inizializzazioni(int trigger) {
     int caso;
     int ritorno=0;
     int esci=0;
```

```
char *appoggio=NULL;

if(chiama_funz_errore==0) {
      chiama_funz_errore=1;
      ritorno = 0;
      return ritorno;
}

if(trigger==0) {
      vett_sx=stable->nome;
      info_sx=stable->info;
      appoggio=strchr(vett_sx,'*');
      if(appoggio==NULL)
            caso=0;
      else
            caso=1;
}
else {
      rec=stable;
      while(rec!=NULL && esci==0) {
            if(strcmp(rec->nome,q)==0) {

                  if(match(rec->info,"^.*property.*$")==1)
                        caso=1;
                  else
                        caso=0;

                  esci=1;
                  vett_sx=q;
            }
            else if(strcmp(rec->nome,q1)==0) {
                  caso=1;
                  esci=1;
                  vett_sx=q1;
            }
            rec=rec->prox;
      }
      if(caso==0) {
            info_sx=info_from_simbolo(vett_sx);
            if(info_sx==NULL) {
                  ritorno = 4;
                  return ritorno;
            }
      }
      else {
            info_sx=info_from_simbolo(vett_sx);

            if(info_sx==NULL) {
                ritorno = 5;
                return ritorno;
            }
      }
}

if(caso==0) {
      if(flag_nome_funz==1) {
            strcpy(p,n);
```

```
            flag_nome_funz==0;
            num_argomenti=1;
      }

      info_dx=info_from_simbolo(p);
      if(info_dx!=NULL) {
            if(strcmp(info_dx,info_sx)!=0) {
                  ritorno = 1;
                  return ritorno;
            }
      }
      else {
            p1=(char *)malloc(strlen(p)+2);
            strcpy(p1,p);
            strcat(p1,"*");
            info_dx=info_from_simbolo(p1);
            if(info_dx!=NULL) {
                  ritorno = 6;
                  return ritorno;
            }
            else {
                  strcpy(provv,p);
                  strcat(provv,"_nome_funz");
                  info_dx=info_from_simbolo(provv);
                  if(info_dx!=NULL) {
                        if(strcmp(info_dx,info_sx)!=0) {
                              ritorno = 1;
                              return ritorno;
                        }
                  }
                  else {
                        strcat(provv,"*");
                        info_dx=info_from_simbolo(provv);
                        if(info_dx!=NULL) {
                              ritorno = 7;
                              return ritorno;
                        }
                        else {
                              ritorno = -1;
                              return ritorno;
                        }
                  }
            }
      }
}
else {
      if(flag_nome_funz==1) {
            strcpy(p,n);
            flag_nome_funz==0;
      }
      if(null_flag==1)
      {
            null_flag=0;
            ritorno=0;
            return ritorno;
      }
      p1=(char *)malloc(strlen(p)+2);
```

```
	strcpy(p1,p);
	strcat(p1,"*");
	info_dx=info_from_simbolo(p1);

	if(info_dx!=NULL) {
		if(strcmp(info_dx,info_sx)!=0) {
			if(match(info_sx,"^.*property.*$")==0) {
				ritorno = 2;
				return ritorno;
			}
		}
	}
	else {
		a=info_from_simbolo(p);
		if(a!=NULL) {
			if(match(a,"^.*property.*$")==0) {
				ritorno = 3;
				return ritorno;
			}
			else {
				ritorno = 0;
				return ritorno;
			}
		}
		else {
			strcpy(provv,p);
			strcat(provv,"_nome_funz");

			a=info_from_simbolo(provv);

			if(a!=NULL) {
				ritorno = 8;
				return ritorno;
			}
			else {
				strcat(provv,"*");
				a=info_from_simbolo(provv);
				if(a!=NULL) {
					if(strcmp(a,info_sx)==0){
						ritorno = 0;
						return ritorno;
					}
					else {
						ritorno = 2;
						return ritorno;
					}
				}
			}

		}
		ritorno = -2;
		return ritorno;

	}
}

return 0;}
```

La funzione si occupa della gestione degli errori in inizializzazioni ed assegnamenti. Il trigger passato come argomento serve a discriminare tra i due casi, infatti nel primo è sufficiente assegnare l'elemento in testa alla Symbol Table alle variabili contenenti le informazioni sulla parte sinistra dell'espressione poiché l'inizializzazione è un'istruzione del tipo:

int a = "inizializzazione";

e quindi "int a" è l'ultimo elemento inserito nella tabella.

Sia che si tratti di assegnamento che di inizializzazione occorre capire se ci si trova a controllare la semantica di variabili puntatore o no. Dato che il nome di un puntatore viene memorizzato nella tabella con il suffisso '*' è opportuno avere coscienza di cosa deve essere precisamente ricercato.

Particolare attenzione viene prestata alle istruzioni del tipo:

int a=funzione();

b=funzione(); /* con b dichiarata in precedenza */

perché in questo caso è necessario ricercare nella tabella dei simboli un record il cui nome presenti il suffisso "_nome_funz" oppure "_nome_funz*".

Eventuali errori fra i tipi delle variabili o fra questi ed i tipi di ritorno delle funzioni vengono rilevati a run-time tramite controlli implementati con le string compare. È opportuno considerare che in objective C è possibile assegnare una property ad una variabile puntatore, pertanto si evince la necessità di un controllo su istruzioni del tipo:

NSString *str = classe.proprieta;

classe.proprieta = puntatore;

implementato tramite la direttiva:

match(rec->info,"^.*property.*$")

opportunamente inserita fra le string compare.

```
void push_token(char *token_type, char *token_name, int token_id)
{
 struct token_stack *s;
 s=(struct token_stack*)malloc(sizeof(struct token_stack));
 s->toke = (char *)malloc(strlen(token_type)+1);
```

```
 s->tname = (char *)malloc(strlen(token_name)+1);

 strcpy(s->toke,token_type);
 strcpy(s->tname,token_name);
 s->ident=token_id;
 s->prox=ts_head;
 ts_head = s;

}

void flush_token_stack()
{
 struct token_stack *f=ts_head;
 if(ts_head!=NULL) {
       while(ts_head->prox!= NULL) {
             f=ts_head->prox;
             free(ts_head);
             ts_head=f;
       }
       free(ts_head);
       ts_head=NULL;
 }

}
```

Durante la stesura del codice, si è evinta la necessità di implementare un'ulteriore pila contenente informazioni riguardanti gli ultimi token passati dallo scanner al parser allo scopo, ad esempio, di tenere traccia degli argomenti di una funzione.

In questo caso, come in altri omologhi, non è possibile attendere che il parser riduca l'intera regola per manipolare le informazioni sui dati che nel frattempo ha elaborato (la variabile p, infatti, contiene solo l'ultimo token riconosciuto), quindi è necessario servirsi di questo stack per non perdere informazioni sui primi n-1 argomenti della funzione. Nel momento in cui lo scanner riconosce alcuni token invoca la funzione "push_token()" specificando nome, tipo ed id del token stesso; quando viene ridotta la regola corrispondente alla definizione di una funzione la pila viene svuotata tramite la chiamata a "flush_token_stack()".

```
void carica_stable(char *elenco)
{
 FILE *fp=NULL;
 FILE *fst=NULL;
 char buffer[512];
 char entry[512];
 int i,j,flag=0;
 int position;
 char *simb;
 char *inf;
```

```
 record * p;

 printf("Inizio caricamento Symbol table.\n");
 fp=fopen(elenco,"r");
 if(fp==NULL)
 {
        printf("file non trovato \n");
 }
 while(!feof(fp))
 {

       fscanf(fp,"%s",buffer);
       fst=fopen(buffer,"r");
       if(fst==NULL)
       {
             printf("impossibile aprire il file\n");
       }
       while(!feof(fst))
       {
             fgets(entry, 512, fst);
             flag=0;
             for(i=0;i<512 && entry[i]!='\n' && flag==0;i++)
             {
                   if(entry[i]=='\t' || entry[i]==':')
                   {
                         position=i;
                         flag=1;
                   }
             }

             simb = malloc(position+1);
             inf = malloc(512-position);
             for(i=0;i<position;i++)
             {
                 simb[i]=entry[i];
             }
             entry[position]='\0';

             j=0;
             for(i=position+1;i<512;i++)
             {
                   inf[j] = entry[i];
                   j++;
             }
             inf[j]='\0';

             putsimbolo(simb,inf);
       }

 }
 printf("0....25....50....75....100\n");
 printf("Symbol table caricata.\n");

}
```

La funzione si occupa del caricamento preliminare della tabella dei simboli attingendo le informazioni da N files di testo elencati nel file di elenco. Il path dell'elenco è l'unico argomento che questa funzione prende in input. Ognuno dei file da processare ha una struttura del tipo:

nome1 tipo1

nome2 tipo2

(…)

nomeM tipoM

dove i tipi corrispondono a quelli specifici dell'objective C.

La sezione *dichiarazioni*:

```
%expect 200
%error-verbose
%locations

%token IDENTIFIER
%token SIZEOF
%token STRING_LITERAL
%token CONSTANT

%token HEX
%token INTEGER

%token PTR_OP
%token INC_OP
%token DEC_OP
%token LEFT_OP
%token RIGHT_OP
%token LE_OP
%token GE_OP
%token EQ_OP
%token NE_OP

%token AND_OP
%token OR_OP
%token MUL_ASSIGN
%token DIV_ASSIGN
%token MOD_ASSIGN
%token ADD_ASSIGN

%token SUB_ASSIGN
%token LEFT_ASSIGN
%token RIGHT_ASSIGN
%token AND_ASSIGN
```

```
%token XOR_ASSIGN
%token OR_ASSIGN
%token TYPE_NAME


%token TYPEDEF
%token EXTERN
%token STATIC
%token AUTO
%token REGISTER

%token CHAR
%token SHORT
%token INT
%token LONG
%token SIGNED
%token UNSIGNED
%token FLOAT
%token DOUBLE
%token VOLATILE
%token VOID
%token CONST

%token STRUCT
%token UNION
%token ENUM
%token ELLIPSIS

%token CASE
%token DEFAULT
%token IF
%token ELSE
%token SWITCH
%token WHILE
%token DO
%token FOR
%token GOTO
%token CONTINUE
%token BREAK
%token RETURN

%token DECLSPEC
%token DLLIMPORT
%token DLLEXPORT

%token INTERFACE
%token IMPLEMENTATION
%token PROTOCOL
%token END
%token CLASS
%token PRIVATE
%token PUBLIC
%token PROTECTED
%token OBJC_ATCLASS
%token OBJC_INTERFACE
%token OBJC_PROP
```

```
%token OBJC_END
%token OBJC_IMPLEMENTATION
%token OBJC_SYNTH

%token OBJC_NONATOMIC
%token OBJC_RETAIN
%token OBJC_COPY
%token OBJC_ASSIGN
%token OBJC_READWRITE
%token OBJC_READONLY
%token OBJC_SUPER
%token OBJC_RELEASE
%token OBJC_ALLOC
%token OBJC_DEALLOC
%token OBJC_SELF
%token OBJC_THIS
%token OBJC_BOOL
%token OBJC_YES
%token OBJC_NO
%token OBJC_NULL
%token OBJC_NIL
%token OBJC_INIT

%token OBJC_DATA_TYPE
%token OBJC_DEF
%token OBJC_CLASS_NAME
%token OBJC_VARI
%token OBJC_CLASS_METHOD
%token OBJC_CONSTANT
%token OBJC_DELEGATE_METHOD
%token OBJC_FUNCTION
%token OBJC_INSTANCE_PROPERTY
%token OBJC_PROTOCOL_METHOD
%token OBJC_INSTANCE_METHOD


%start vamos_a_pensar
```

La sezione contiene la definizione dei token che il parser è in grado di riconoscere. È necessario che nel funzionamento combinato di Flex e Bison vi siano tante direttive quanti sono i token che lo scanner può passare al parser e che i nomi che li identificano corrispondano poiché, in maniera trasparente al programmatore, i token vengono enumerati e viene definita una macro per ognuno di essi. Nella pratica, quindi, è come se si utilizzassero dei numeri interi di riferimento.

La direttiva:

- %error-verbose consente la visualizzazione degli errori direttamente sullo stream di output quando viene richiamata la funzione yyerror()

- %locations permette di ottenere maggiori informazioni circa l'errore riscontrato, ad esempio consente di far stampare a video anche il token che ci si aspetterebbe di ricevere
- %expect permette di ignorare i warning relativi a conflitti del tipo shift/reduce, dato che sono conflitti che non sempre portano ad errori nelle grammatiche e che potrebbero essere risolti se si disponesse di un maggior numero di lookahead tokens
- %start nodo specifica l'assioma della grammatica. Nel caso di blocK questo prende il nome di vamos_a_pensar.

La sezione *regole grammaticali*:

```
espressione_base
: IDENTIFIER
| CONSTANT {constant_flag=1;}
| STRING_LITERAL
| '(' espressione ')'
| objc_aggiunte_espressione
;

objc_aggiunte_espressione
: OBJC_YES {boolean_flag=1;}
| OBJC_NO {boolean_flag=1;}
| OBJC_NULL{null_flag=1;}
| OBJC_SELF
| OBJC_NIL
| OBJC_THIS
;
```

"Objc_aggiunte_espressione" estende i nodi foglia afferenti ad un'espressione base facendo sì che il parser riconosca parole chiave dell'objc.

```
espressione_memoria_dinamica_objc
: espressione_allocazione
| espressione_release_retain
;

espressione_allocazione
: '[' objc_specificatore_tipo OBJC_ALLOC ']' {
    chiama_funz_errore = 0;
    if(q!=NULL && flag_alloc==0) {
     if(strcmp(p,q)!=0) {
          sprintf(provv,"\nErrore allocazione %s = %s, errore alla riga
%d\n",q,p,numeroriga);
          yyerror(provv);
     }
```

```
     }
     else {
      flag_alloc=0;
      sprintf(provv,"\nErrore allocazione, si vuole allocare %s che non è
un puntatore, errore alla riga %d\n",p,numeroriga);
      yyerror(provv);
     }
}
| '[' objc_specificatore_tipo OBJC_ALLOC ']' lista {
     chiama_funz_errore=0;
     struct token_stack *tt=ts_head;
     while(tt->prox->prox->prox!=NULL) {
      tt=tt->prox;
     }
     strcpy(sp_tipo,tt->tname);
     tt=ts_head;
     while(tt->prox!=NULL)
     {
      tt=tt->prox;
     }
     if(strcmp(sp_tipo,tt->tname)!=0) {
      sprintf(provv,"\nErrore allocazione %s = %s, errore alla riga
%d\n",tt->tname,sp_tipo,numeroriga);
      yyerror(provv);
     }
}
| nodolo {
     chiama_funz_errore = 0;
}
;
```

L' "espressione_allocazione" riduce espressioni che coinvolgono la memoria dinamica e pertanto si occupa di gestire eventuali errori di incompatibilità di tipo.

```
nodolo
: '[' espressione_allocazione ']'
| nodolo '[' espressione_allocazione ']'
| espressione_allocazione OBJC_PROTOCOL_METHOD
;

lista
: lista_init
| lista_quadre
| lista lista_quadre
| lista lista_init
;

lista_init
: OBJC_INIT
| init_format
| lista_init ',' init_format
;
```

```
init_format
: espressione_base
| espressione_base ':' espressione_base
;

espressione_release_retain
: selettore_prop OBJC_RELEASE
| selettore_prop OBJC_RETAIN
;
```

Il nodo provvede al parsing di espressioni del tipo [oggetto release/retain].

```
espressione_postfix
: espressione_base
| espressione_postfix '[' espressione ']'
| espressione_postfix '(' {strcpy(n,ts_head->tname);} ')'
{flag_nome_funz=1;}
| espressione_postfix '(' {strcpy(n,ts_head->tname);}
lista_argomenti_espressione ')' {flag_nome_funz=1;}
| espressione_postfix '.' selettore_prop
| espressione_postfix PTR_OP IDENTIFIER
| espressione_postfix INC_OP
| espressione_postfix DEC_OP
| espressione_memoria_dinamica_objc
| '[' espressione_memoria_dinamica_objc ']'
| '[' espressione_super OBJC_INIT']'
;

espressione_super
: objc_s_p
| espressione_super objc_s_p
;

objc_s_p
: OBJC_SUPER
;
```

Il nodo permette il riconoscimento del costrutto [super nome_classe].

```
lista_argomenti_espressione
: espressione_assegnamento
| lista_argomenti_espressione ',' {num_argomenti++;}
espressione_assegnamento
;

espressione_unaria
: espressione_postfix
| INC_OP espressione_unaria
| DEC_OP espressione_unaria
| operatore_unario espressione_cast
| SIZEOF espressione_unaria
| SIZEOF '(' nome_tipo ')' ;
```

```
operatore_unario
: '&'
| '*'
| '+'
| '-'
| '~'
| '!'
| '/'
;

espressione_cast
: espressione_unaria
| '(' nome_tipo ')' espressione_cast
;

espressione_moltiplicativa
: espressione_cast
| espressione_moltiplicativa '*' espressione_cast
| espressione_moltiplicativa '/' espressione_cast
| espressione_moltiplicativa '%' espressione_cast
;

espressione_addizione
: espressione_moltiplicativa
| espressione_addizione'+' espressione_moltiplicativa
| espressione_addizione '-' espressione_moltiplicativa
;

espressione_shift
: espressione_addizione
| espressione_shift LEFT_OP espressione_addizione
| espressione_shift RIGHT_OP espressione_addizione
;

espressione_relazionale
: espressione_shift
| espressione_relazionale '<' espressione_shift
| espressione_relazionale  '>' espressione_shift
| espressione_relazionale LE_OP espressione_shift
| espressione_relazionale GE_OP espressione_shift
;

espressione_uguaglianza
: espressione_relazionale
| espressione_uguaglianza EQ_OP espressione_relazionale
| espressione_uguaglianza NE_OP espressione_relazionale
;

espressione_and
: espressione_uguaglianza
| espressione_and '&' espressione_uguaglianza
;

espressione_or_esclusivo
: espressione_and
| espressione_or_esclusivo '^' espressione_and
;
```

```
espressione_or_inclusivo
: espressione_or_esclusivo
| espressione_or_inclusivo '|' espressione_or_esclusivo
;

espressione_and_logico
: espressione_or_inclusivo
| espressione_and_logico AND_OP espressione_or_inclusivo
;

espressione_or_logico
: espressione_and_logico
| espressione_or_logico OR_OP espressione_and_logico
;

espressione_condizionale
: espressione_or_logico
| espressione_or_logico '?' espressione ':' espressione_condizionale
;
```

Questa serie di nodi viene richiamata partendo da "espressione_assegnamento" fino ad un'espressione basilare. L'albero, in questo caso, viene risalito in profondità e non in ampiezza dato che, se si entra in "espressione_assegnamento", si troverà obbligatoriamente almeno un nodo che identifichi l'espressione dell'input evitando quindi situazioni di loop.

```
espressione_assegnamento
: espressione_condizionale
| espressione_unaria
{
     q=(char *)malloc(strlen(p)+2);
     strcpy(q,p);
     q1=(char *)malloc(strlen(p)+2);
     strcpy(q1,p);
     strcat(q1,"*");
}
operatore_assegnamento espressione_assegnamento
{
     tipo_errore = errore_tipo_inizializzazioni(1);
     if(tipo_errore==1) {
      sprintf(provv,"\nErrore tipi variabile a dx e sx dell'assegnamento,
probabilmente sarebbe opportuno controllare la riga %d\n",numeroriga);
      yyerror(provv);
     }
     else if(tipo_errore==-1) {
      sprintf(provv,"\nVariabile %s non presente in ST, errore alla riga
%d\n",p,numeroriga);
      yyerror(provv);
     }
     else if(tipo_errore==2) {
```

```
      sprintf(provv,"\nErrore tipi puntatori a dx e sx
dell'inizializzazione, probabilmente sarebbe opportuno controllare la
riga %d\n",numeroriga);
      yyerror(provv);
     }
     else if(tipo_errore==-2) {
      sprintf(provv,"\nVariabile puntatore %s non presente in ST, errore
alla riga %d\n",p,numeroriga);
      yyerror(provv);
     }
     else if(tipo_errore==3) {
      sprintf(provv,"\nVariabile %s non compatibile nell'assegnamento,
errore alla riga %d\n",p,numeroriga);
      yyerror(provv);
     }
     else if(tipo_errore==4) {
      sprintf(provv,"\nVariabile %s, non presente in ST, errore alla riga
%d\n",q,numeroriga);
      yyerror(provv);
     }
     else if(tipo_errore==5) {
      sprintf(provv,"\nVariabile puntatore %s, non presente in ST, errore
alla riga %d\n",q,numeroriga);
      yyerror(provv);
     }
     else if(tipo_errore==6) {
      sprintf(provv,"\nVariabile puntatore %s, non presente in ST, errore
alla riga %d\n",q,numeroriga);
      yyerror(provv);
     }
     else if(tipo_errore==7) {
      sprintf(provv,"\nSi sta assegnando alla variabile %s il tipo di
ritorno della funzione %s che è un puntatore, errore alla riga
%d\n",q,p,numeroriga);
      yyerror(provv);
     }
     else if(tipo_errore==8) {
      sprintf(provv,"\n Alla variabile puntatore %s si sta assegnando il
tipo di ritorno della funzione %s che non lo è, errore alla riga
%d\n",q,p,numeroriga);
      yyerror(provv);
     }
}
;
```

Questo nodo richiama la funzione di errore e, a seconda del valore che essa restituisce, chiama yyerror() stampando un opportuno messaggio.

```
operatore_assegnamento
: '='
| MUL_ASSIGN
| DIV_ASSIGN
| MOD_ASSIGN
| ADD_ASSIGN
| SUB_ASSIGN
```

```
| LEFT_ASSIGN
| RIGHT_ASSIGN
| AND_ASSIGN
| XOR_ASSIGN
| OR_ASSIGN
;

espressione
: espressione_assegnamento
| espressione ',' espressione_assegnamento
;

espressione_costante
: espressione_condizionale
;

dichiarazione
: specificatori_dichiarazione ';'
| dichiarazione_tipo ';'
| specificatori_dichiarazione lista_dichiaratore_iniz ';' {
    flush_token_stack();
    conta_dichiaratore=0;
}
| error ';' { printf(".\n"); }
| error '}' { printf(".\n"); }
| error { printf(".\n"); }
;
```

In questo nodo è presente il token speciale error che consente al parser di continuare la propria esecuzione anche dopo la chiamata di yyerror().
La scrittura “error nome_token” fa scartare l’input da dove si è generato l’errore fino a nome_token.

```
specificatori_dichiarazione
: specificatore_classe_storage
| specificatore_classe_storage specificatori_dichiarazione
| specificatore_tipo
| specificatore_tipo specificatori_dichiarazione
| qualificatore_tipo
| qualificatore_tipo specificatori_dichiarazione
| specdich specificatore_classe_storage
| specdich specificatore_classe_storage specificatori_dichiarazione
| specdich specificatore_tipo
| specdich specificatore_tipo specificatori_dichiarazione
| specdich qualificatore_tipo
| specdich qualificatore_tipo specificatori_dichiarazione
;

lista_dichiaratore_iniz
: dichiaratore_iniz
| lista_dichiaratore_iniz ',' dichiaratore_iniz
;

dichiaratore_iniz
```

```
: dichiaratore
| dichiaratore '=' inizializzatore
;

tipo_specdich
: DLLIMPORT
| DLLEXPORT
;

specdich
: DECLSPEC '(' tipo_specdich ')'
;

specificatore_classe_storage
: EXTERN
| STATIC
| AUTO
| REGISTER
;

tipo_dichiaratore
: puntatore dichiaratore_diretto_tipo
| dichiaratore_diretto_tipo
;

dichiaratore_diretto_tipo
: IDENTIFIER    {
     if(controlla (p)==0)
      putsimbolo(p,"dato utente");
      }
| '(' tipo_dichiaratore ')'
| dichiaratore_diretto_tipo '[' espressione_costante ']'
| dichiaratore_diretto_tipo'[' ']'
| dichiaratore_diretto_tipo '(' lista_tipo_parametri ')'
| dichiaratore_diretto_tipo '(' lista_identificatori ')'
| dichiaratore_diretto_tipo '(' ')'
;

dichiarazione_tipo
: TYPEDEF specificatori_dichiarazione tipo_dichiaratore
;

specificatore_tipo
: VOID
| CHAR
| SHORT
| INT
| LONG
| FLOAT
| DOUBLE
| SIGNED
| UNSIGNED
| specificatore_struct_o_union
| specificatore_enum
| TYPE_NAME
| objc_specificatore_tipo
;
```

```
objc_specificatore_tipo
: OBJC_DATA_TYPE
| OBJC_CLASS_NAME
| OBJC_BOOL
| OBJC_CONSTANT
;

specificatore_struct_o_union
: struct_o_union IDENTIFIER '{' lista_dichiarazione_struct '}'
{strcpy(provv,"struct "); strcat(provv,p); controlla_aggiungi(p,provv);}
| struct_o_union '{' lista_dichiarazione_struct '}'
| struct_o_union IDENTIFIER {strcpy(provv,"struct "); strcat(provv,p);
controlla_aggiungi(p,provv);}
;

struct_o_union
: STRUCT
| UNION
;

lista_dichiarazione_struct
: dichiarazione_struct
| lista_dichiarazione_struct dichiarazione_struct
;

dichiarazione_struct
: lista_specificatore_qualificatore lista_dichiaratore_struct ';'
;

lista_specificatore_qualificatore
: specificatore_tipo lista_specificatore_qualificatore
| specificatore_tipo
| qualificatore_tipo lista_specificatore_qualificatore
| qualificatore_tipo
;

lista_dichiaratore_struct
: dichiaratore_struct
| lista_dichiaratore_struct ',' dichiaratore_struct
;

dichiaratore_struct
: dichiaratore
| ':' espressione_costante
| dichiaratore ':' espressione_costante
;

specificatore_enum
: ENUM '{' lista_enum '}'
| ENUM IDENTIFIER '{' lista_enum '}' { controlla_aggiungi(p,"dato
utente");}
| ENUM IDENTIFIER { controlla_aggiungi(p,"dato utente");}
;

lista_enum
: enumeratore
```

```
| lista_enum ',' enumeratore
;

enumeratore
: IDENTIFIER
| IDENTIFIER '=' espressione_costante
;

qualificatore_tipo
: CONST
| VOLATILE
;

dichiaratore
: puntatore {flag_funzione_puntatore=1;} dichiaratore_diretto {
    struct token_stack *z=ts_head;
    if(ts_head!=NULL) {
     z=ts_head->prox;
     if(z!=NULL) {
           strcpy(provv,z->toke);
           strcat(p,"*");
           if(r!=NULL) {
                 q=(char *)malloc(strlen(r)+1);
                 strcpy(q,r);
           }
           controlla_aggiungi(p,provv);
           flag1++;
     }
    }

}
| dichiaratore_diretto  {   struct token_stack *z=ts_head;

    if(conta_dichiaratore!=0) {
     ts_head=z->prox;
     free(z);
    }

    if(ts_head!=NULL) {
     z=ts_head->prox;
     if(z!=NULL) {
           strcpy(provv,z->toke);
           if(r!=NULL) {
                 flag_alloc=1;
           }
           controlla_aggiungi(p,provv);
     }
    }
    conta_dichiaratore++;
    flag1++;

}
;
```

Il nodo provvede all'inserimento in symbol table di nuove variabili dichiarate dall'utente. Nel caso di puntatori si procede inserendo un "*" come suffisso del nome di quest' ultime.

```
dichiaratore_diretto
:  selettore_prop
| '(' dichiaratore ')'
| dichiaratore_diretto '[' espressione_costante ']'
| dichiaratore_diretto '[' ']'
| dichiaratore_diretto '(' {
     struct token_stack *i=ts_head;
     for(i=ts_head;i->prox!=NULL;i=i->prox){
     }
     strcpy(provv,i->toke);
     strcat(p,"_nome_funz");
     if(flag_funzione_puntatore==1) {
      strcat(p,"*");
      flag_funzione_puntatore=0;
     }
     controlla_aggiungi(p,provv);
} lista_tipo_parametri
{conta_dichiaratore=0;}')'{flush_token_stack();conta_dichiaratore=0;flag
_funzione_puntatore=0;}
| dichiaratore_diretto '(' {
     struct token_stack *i=ts_head;
     for(i=ts_head;i->prox!=NULL;i=i->prox){
     }
     strcpy(provv,i->toke);
     strcat(p,"_nome_funz");
     if(flag_funzione_puntatore==1) {
      strcat(p,"*");
      flag_funzione_puntatore=0;
     }
     controlla_aggiungi(p,provv);

}lista_identificatori ')'
{flush_token_stack();conta_dichiaratore=0;flag_funzione_puntatore=0;}
| dichiaratore_diretto '(' {

     struct token_stack *i=ts_head;
     for(i=ts_head;i->prox!=NULL;i=i->prox){
     }

     strcpy(provv,i->toke);
     strcat(p,"_nome_funz");
     if(flag_funzione_puntatore==1) {
      strcat(p,"*");
      flag_funzione_puntatore=0;
     }
     controlla_aggiungi(p,provv);

}')'
{flush_token_stack();conta_dichiaratore=0;flag_funzione_puntatore=0;}
;
```

Il nodo provvede all'inserimento in symbol table di funzioni dichiarate dall'utente, inserendo il suffisso "_nome_funz" al nome definito e un "*" nel caso in cui esse abbiano un puntatore come valore di ritorno.

```
puntatore
: '*'
| '*' lista_tipo_qualificatore
| '*' puntatore
| '*' lista_tipo_qualificatore puntatore
;

lista_tipo_qualificatore
: qualificatore_tipo
| lista_tipo_qualificatore qualificatore_tipo
;


lista_tipo_parametri
: lista_parametri
| lista_parametri ',' ELLIPSIS
;

lista_parametri
: dichiazione_parametri
| lista_parametri ','dichiazione_parametri
;

dichiazione_parametri
: specificatori_dichiarazione dichiaratore {flush_token_stack();
conta_dichiaratore=0;}
| specificatori_dichiarazione dichiaratore_astratto
| specificatori_dichiarazione
;

lista_identificatori
: IDENTIFIER
| lista_identificatori ',' IDENTIFIER
;

nome_tipo
: lista_specificatore_qualificatore
| lista_specificatore_qualificatore dichiaratore_astratto
;

dichiaratore_astratto
: puntatore {flag_funzione_puntatore = 1;}
| dichiaratore_astratto_diretto
| puntatore dichiaratore_astratto_diretto
;

dichiaratore_astratto_diretto
: '(' dichiaratore_astratto ')'
| '[' ']'
| '[' espressione_costante ']'
```

```
| dichiaratore_astratto_diretto '[' ']'
| dichiaratore_astratto_diretto '[' espressione_costante ']'
| '(' ')'
| '(' lista_tipo_parametri ')'
| dichiaratore_astratto_diretto '(' ')'
| dichiaratore_astratto_diretto '(' lista_tipo_parametri ')'
;

inizializzatore
: espressione_assegnamento {
    tipo_errore = errore_tipo_inizializzazioni(0);
    if(tipo_errore==1) {
     sprintf(provv,"\nErrore tipi variabile a dx e sx
dell'inizializzazione, probabilmente sarebbe opportuno controllare la
riga %d\n",numeroriga);
     yyerror(provv);
    }
    else if(tipo_errore==-1) {
     sprintf(provv,"\nVariabile %s non presente in ST, errore alla riga
%d\n",p,numeroriga);
     yyerror(provv);
    }
    else if(tipo_errore==2) {
     sprintf(provv,"\nErrore tipi puntatori a dx e sx
dell'inizializzazione, probabilmente sarebbe opportuno controllare la
riga %d\n",numeroriga);
     yyerror(provv);
    }
    else if(tipo_errore==-2) {
     sprintf(provv,"\nVariabile puntatore %s non presente in ST, errore
alla riga %d\n",p,numeroriga);
     yyerror(provv);
    }
    else if(tipo_errore==3) {
     sprintf(provv,"\nVariabile %s non compatibile
nell'inizializzazione, errore alla riga %d\n",p,numeroriga);
     yyerror(provv);
    }
    else if(tipo_errore==7) {
     sprintf(provv,"\nSi sta inizializzando una variabile con il tipo di
ritorno della funzione %s che è un puntatore, errore alla riga
%d\n",p,numeroriga);
     yyerror(provv);
    }
    else if(tipo_errore==8) {
     sprintf(provv,"\n Si sta inizializzando una variabile puntatore con
il tipo di ritorno della funzione %s che non lo è, errore alla riga
%d\n",p,numeroriga);
     yyerror(provv);
    }
}
| '{' lista_inizializzatore '}'
| '{' lista_inizializzatore ',' '}'
;
```

Il nodo “inizializzatore” richiama la funzione definita in precedenza e stampa un opportuno messaggio in base al codice che essa restituisce.

```
lista_inizializzatore
: inizializzatore
| lista_inizializzatore ',' inizializzatore
;

istruzione
: istruzione_etichettata {constant_flag=0;null_flag=0;}
| istruzione_composta {constant_flag=0; null_flag=0;}
| istruzione_espressione {constant_flag=0; null_flag=0;}
| istruzione_selezione {constant_flag=0; null_flag=0;}
| istruzione_iterazione {constant_flag=0; null_flag=0;}
| istruzione_jump {null_flag=0;}
| istruzione_OBJC {constant_flag=0;null_flag=0;}
;

istruzione_OBJC
: '[' OBJC_SUPER selettore ']' ';'
| '[' OBJC_SUPER selettore ':' selettore_prop ']' ';'
| '[' lista ']' ';'
;
```

Questo nodo contiene regole per derivare alcune espressioni dell’objc che non rientrano fra quelle della grammatica del C.

```
lista_quadre
: objc_espr_quadre
| lista_quadre objc_espr_quadre
;


objc_espr_quadre
: selettore_prop2
| selettore_prop2 ':' selettore_prop
| objc_espr_quadre '.' selettore_prop
;

istruzione_etichettata
: IDENTIFIER ':' istruzione
| CASE espressione_costante ':' istruzione
| DEFAULT ':' istruzione
;

istruzione_composta
: '{' '}'
| '{' lista_istruzioni '}'
| '{' lista_dichiarazioni '}'
| '{' lista_dichiarazioni lista_istruzioni '}'
| '{' lista_istruzioni lista_dichiarazioni '}'
;
```

```
lista_dichiarazioni
: dichiarazione
| lista_dichiarazioni dichiarazione
;

lista_istruzioni
: istruzione
| lista_istruzioni istruzione
;

istruzione_espressione
: ';' {flag_nome_funz=0;}
| espressione ';' {flag_nome_funz=0;}
;

istruzione_selezione
: IF '(' espressione ')' istruzione
| IF '(' espressione ')' istruzione ELSE istruzione
| SWITCH '(' espressione ')' istruzione
;

istruzione_iterazione
: WHILE '(' espressione ')' istruzione
| DO istruzione WHILE '(' espressione ')' ';'
| FOR '(' istruzione_espressione istruzione_espressione')' istruzione
| FOR '(' istruzione_espressione istruzione_espressione espressione ')'
istruzione
;

istruzione_jump
: GOTO IDENTIFIER ';' {constant_flag=0;}
| CONTINUE ';' {constant_flag=0;}
| BREAK ';' {constant_flag=0;}
| RETURN ';' {constant_flag=0; rec=stable; int flag=0;
    while(flag==0 && rec!=NULL)
    {
     if(match(rec->nome,"^.*.nome.funz.*$")==1)
     {
          flag=1;
          if(strcmp(rec->info,"void")!=0)
          {
               sprintf(provv,"Errore incompatibilità tipo la funzione
prevede %s e invece non restituisce alcun valore, errore alla riga
%d\n",rec->info,numeroriga);
               yyerror(provv);
          }
     }else
          rec=rec->prox;
    }

    returnflag=1;
    constant_flag=0;
}
| RETURN espressione {

    int flag=0;
```

```
    int fdg=0;
    char *appoggio=NULL;
    rec=stable;
    while(flag==0 && rec!=NULL)
    {
     if(match(rec->nome,"^.*.nome.funz.*$")==1)
     {
           flag=1;

           if(boolean_flag==0 && constant_flag==0 && null_flag==0) {
                 fdg=1;
                 appino=info_from_simbolo(p);
                 if(appino==NULL)
                 {
                       strcpy(provv,p);
                       strcat(provv,"*");
                       appino=info_from_simbolo(provv);
                       if(appino==NULL)
                       {
                             if(constant_flag==0) {
                                   sprintf(provv,"Errore, la funzione
restituisce %s, variabile non dichiarata in precedenza, errore alla riga
%d\n",p,numeroriga);
                                   yyerror(provv);
                             }
                       }
                       else{
                             appoggio=strchr(rec->nome,'*');
                             if(appoggio==NULL) {
                                   sprintf(provv,"Errore, la funzione
restituisce un puntatore ad %s, invece dovrebbe restituire un %s, errore
alla riga %d\n",appino,rec->info,numeroriga);
                                   yyerror(provv);
                             }
                             else {
                                   if(strcmp(appino,rec->info)!=0) {
                                         sprintf(provv,"Errore, la funzione
restituisce %s, invece dovrebbe restituire un puntatore ad %s, errore
alla riga %d\n",appino,rec->info,numeroriga);
                                         yyerror(provv);
                                   }
                             }
                       }
                 }
                 else{
                       appoggio=strchr(rec->nome,'*');
                       if(appoggio!=NULL) {
                             sprintf(provv,"Errore, la funzione restituisce
%s, invece dovrebbe restituire un puntatore ad %s, errore alla riga
%d\n",p,rec->info,numeroriga);
                             yyerror(provv);
                       }
                       else {
                             if(strcmp(appino,rec->info)!=0) {
                                   sprintf(provv,"Errore, la funzione
restituisce %s, invece dovrebbe restituire %s, errore alla riga
%d\n",p,rec->info,numeroriga);
```

```
                                    yyerror(provv);
                              }
                        }
                  }

            }

      }
      else
            rec=rec->prox;

     }
     if(rec==NULL) {
      sprintf(provv,"Errore, la funzione restituisce %s, variabile non
dichiarata in precedenza, errore alla riga %d\n",p,numeroriga);
      yyerror(provv);
     }

     if(fdg==0 && null_flag==0){
      if(constant_flag==0) {
            if(appino!=NULL){
                  if(flag==1 && strcmp(rec->info,appino)!=0 && strcmp(rec-
>info,"void")!=0 && strcmp(rec->info,"identifier")!=0 &&
boolean_flag==0)
                  {
                        sprintf(provv,"Errore incompatibilità tipo la
funzione restituisce %s e invece prevede %s, errore alla riga
%d\n",appino,rec->info,numeroriga);
                        yyerror(provv);
                  }
                  if(strcmp(rec->info,"void")==0)
                  {
                        sprintf(provv,"Errore incompatibilità tipo la
funzione restituisce %s e invece non prevede valore di ritorno, errore
alla riga %d\n",appino,numeroriga);
                        yyerror(provv);
                  }
            }
      }
      else {
            if((strcmp(rec->info,"void")==0 || strcmp(rec-
>info,"identifier")==0)) {
                  sprintf(provv,"Errore incompatibilità tipo la funzione
restituisce una costante e invece non prevede alcun tipo di ritorno,
errore alla riga %d\n",numeroriga);
                  yyerror(provv);
            }
      }

      if(strcmp(rec->info,"objc_bool")==0 && boolean_flag==0) {
            sprintf(provv,"Errore incompatibilità tipo la funzione
dovrebbe restituire un booleano, errore alla riga %d\n",numeroriga);
            yyerror(provv);
      }
     }
     returnflag=1;
     constant_flag=0;
```

```
    boolean_flag=0;
    flag_funzione_puntatore=0;
    null_flag=0;
} ';'
| RETURN tipi_ritorno_objc {

    int flag=0;
    int fdg=0;
    char *appoggio=NULL;
    rec=stable;
    while(flag==0 && rec!=NULL)
    {
     if(match(rec->nome,"^.*.nome.funz.*$")==1)
     {
          flag=1;

          if(boolean_flag==0 && constant_flag==0 && null_flag==0) {
               fdg=1;
               appino=info_from_simbolo(p);

               if(appino==NULL)
               {
                    strcpy(provv,p);
                    strcat(provv,"*");
                    appino=info_from_simbolo(provv);
                    if(appino==NULL)
                    {
                         if(constant_flag==0) {
                              sprintf(provv,"Errore, la funzione
restituisce %s, variabile non dichiarata in precedenza, errore alla riga
%d\n",p,numeroriga);
                              yyerror(provv);
                         }
                    }
                    else{
                         appoggio=strchr(rec->nome,'*');
                         if(appoggio==NULL) {
                              sprintf(provv,"Errore, la funzione
restituisce un puntatore ad %s, invece dovrebbe restituire un %s, errore
alla riga %d\n",appino,rec->info,numeroriga);
                              yyerror(provv);
                         }
                         else {
                              if(strcmp(appino,rec->info)!=0) {
                                   sprintf(provv,"Errore, la funzione
restituisce %s, invece dovrebbe restituire un puntatore ad %s, errore
alla riga %d\n",appino,rec->info,numeroriga);
                                   yyerror(provv);
                              }
                         }
                    }
               }
               else{
                    appoggio=strchr(rec->nome,'*');
                    if(appoggio!=NULL) {
                         sprintf(provv,"Errore, la funzione restituisce
%s, invece dovrebbe restituire un puntatore ad %s, errore alla riga
```

```
%d\n",p,rec->info,numeroriga);
                              yyerror(provv);
                        }
                        else {
                              if(strcmp(appino,rec->info)!=0) {
                                    sprintf(provv,"Errore, la funzione
restituisce %s, invece dovrebbe restituire %s, errore alla riga
%d\n",p,rec->info,numeroriga);
                                    yyerror(provv);
                              }
                        }
                  }

            }

      }
      else
            rec=rec->prox;

     }
     if(rec==NULL) {
      sprintf(provv,"Errore, la funzione restituisce %s, variabile non
dichiarata in precedenza, errore alla riga %d\n",p,numeroriga);
      yyerror(provv);
     }

     if(fdg==0 && null_flag==0){
      if(constant_flag==0) {
            if(appino!=NULL){
                  if(flag==1 && strcmp(rec->info,appino)!=0 && strcmp(rec-
>info,"void")!=0 && strcmp(rec->info,"identifier")!=0 &&
boolean_flag==0)
                  {
                        sprintf(provv,"Errore incompatibilità tipo la
funzione restituisce %s e invece prevede %s, errore alla riga
%d\n",appino,rec->info,numeroriga);
                        yyerror(provv);
                  }
                  if(strcmp(rec->info,"void")==0)
                  {
                        sprintf(provv,"Errore incompatibilità tipo la
funzione restituisce %s e invece non prevede valore di ritorno, errore
alla riga %d\n",appino,numeroriga);
                        yyerror(provv);
                  }
            }
      }
      else {
            if((strcmp(rec->info,"void")==0 || strcmp(rec-
>info,"identifier")==0)) {
                  sprintf(provv,"Errore incompatibilità tipo la funzione
restituisce una costante e invece non prevede alcun tipo di ritorno,
errore alla riga %d\n",numeroriga);
                  yyerror(provv);
            }
      }
```

```
      if(strcmp(rec->info,"objc_bool")==0 && boolean_flag==0) {
            sprintf(provv,"Errore incompatibilità tipo la funzione
dovrebbe restituire un booleano, errore alla riga %d\n",numeroriga);
            yyerror(provv);
       }
     }
     returnflag=1;
     constant_flag=0;
     boolean_flag=0;
     flag_funzione_puntatore=0;
     null_flag=0;
} ';'
;
```

Questo nodo tratta la delicata questione del return in una funzione, infatti nasce la necessità gestire una serie di errori derivanti da incompatibilità tra il tipo di ritorno della funzione e la variabile/costante specificata nell'input. In primo luogo il codice ricerca il nome della funzione in Symbol table allo scopo di identificarne il tipo di ritorno. Una volta ridotta questa regola, si ha a disposizione la variabile di ritorno che è pronta per essere confrontata. Se si verificano delle condizioni di errore allora viene stampato a video il messaggio corrispondente.

```
tipi_ritorno_objc
: OBJC_CLASS_NAME
| OBJC_CONSTANT
| OBJC_DATA_TYPE
;

vamos_a_pensar
: dichiarazione_esterna
| vamos_a_pensar dichiarazione_esterna
;
```

Questo è l'assioma della grammatica, che richiama ricorsivamente uno o più nodi di tipo "dichiarazione_esterna"

```
dichiarazione_esterna
        : definizione_funzione
        | dichiarazione
        | interfaccia_classe
        | implementazione_classe
        | categoria_interfaccia
        | categoria_implementazione
        | dichiarazione_protocollo
        | lista_dichiarazione_classe
        ;
```

Da “dichiarazione_esterna” si dipartono tutti i possibili rami dell’albero.

```
definizione_funzione
: specificatori_dichiarazione dichiaratore
lista_dichiarazioni{flag_funzione_puntatore=0;} istruzione_composta
{flag_funzione_puntatore=0; dealloc_flag=0; constant_flag=0;
flush_token_stack();}
| specificatori_dichiarazione dichiaratore {conta_dichiaratore=0;}
istruzione_composta
{

     int flag=0;
     rec = stable;
     for(contatore=0;contatore<flag1;contatore++) {
      rec=rec->prox;
     }
     rec = stable;
     while(rec!=NULL && match(rec->nome,"^.*.nome.funz$")==0) {
      rec=rec->prox;
     }

     flag1=0;
     if(returnflag==0) {
      if(match(rec->info,"^.*void.*$")==0) {
            sprintf(provv,"Errore, manca il return\n");
            yyerror(provv);
      }
     }
     else
      returnflag=0;

      rec=stable;

      if(stable!=NULL) {
            if(match(rec->nome,"^.*nome_funz.*$")==1)
                  flag=1;
            while(stable->prox!=NULL && flag==0) {
                  rec=stable->prox;
                  free(stable);
                  stable=rec;
                  if(match(rec->nome,"^.*nome_funz.*$")==1)
                        flag=1;
            }
      }

     flag_funzione_puntatore=0;
     dealloc_flag=0;
     constant_flag=0;
     flush_token_stack();

}
| dichiaratore lista_dichiarazioni istruzione_composta {
     int flag=0;
     if(returnflag==1) {
      sprintf(provv,"Errore incompatibilità tipo la funzione non prevede
```

```
valore di ritorno, errore alla riga %d\n",numeroriga);
      yyerror(provv);
     }
     rec=stable;

     if(stable!=NULL) {
      if(match(rec->nome,"^.*nome_funz.*$")==1)
            flag=1;
      while(stable->prox!=NULL && flag==0) {
            rec=stable->prox;
            free(stable);
            stable=rec;
            if(match(rec->nome,"^.*nome_funz.*$")==1)
                  flag=1;
      }
     }   flag_funzione_puntatore=0;
     dealloc_flag=0;
     constant_flag=0;
     flush_token_stack();
}
| dichiaratore istruzione_composta {
     int flag=0;
     if(returnflag==1) {
      sprintf(provv,"Errore incompatibilità tipo la funzione non prevede
valore di ritorno, errore alla riga %d\n",numeroriga);
      yyerror(provv);
     }
     rec=stable;

     if(stable!=NULL) {
      if(match(rec->nome,"^.*nome_funz.*$")==1)
            flag=1;
      while(stable->prox!=NULL && flag==0) {
            rec=stable->prox;
            free(stable);
            stable=rec;
            if(match(rec->nome,"^.*nome_funz.*$")==1)
                  flag=1;
      }
     }   flag_funzione_puntatore=0;
     dealloc_flag=0;
     constant_flag=0;
     flush_token_stack();
}
;
```

Il nodo “definizione_funzione”, oltre a contenere i non terminali che lo definiscono, prevede che venga eseguito del codice una volta riconosciuta la definizione della funzione all’interno dell’input. Il codice provvede all’aggiunta in Symbol Table della funzione appena definita. Inoltre, qui viene riconosciuto l’errore che si genera se una funzione che non prevede tipo di ritorno presenta un’istruzione di return. Infine vengono resettati eventuali flag, viene effettuata la

flush sulla pila dei token e vengono eliminate dalla symbol table le informazioni relative alle variabili locali.

```
interfaccia_classe
: OBJC_INTERFACE nome_classe istanza_variabili
lista_dichiarazioni_interfaccia END
| OBJC_INTERFACE nome_classe ':' nome_superclasse istanza_variabili
lista_dichiarazioni_interfaccia END
| OBJC_INTERFACE lista_protocollo_riferimento istanza_variabili
lista_dichiarazioni_interfaccia END
| OBJC_INTERFACE nome_classe ':' nome_superclasse
lista_protocollo_riferimento istanza_variabili
lista_dichiarazioni_interfaccia END
| OBJC_INTERFACE nome_classe '{' '}' END
| OBJC_INTERFACE nome_classe ':' nome_superclasse '{' '}' END
;

implementazione_classe
: OBJC_IMPLEMENTATION nome_classe istanza_variabili
{flush_token_stack();conta_dichiaratore=0;}
lista_definizione_implementazioni END
| OBJC_IMPLEMENTATION nome_classe ':' nome_superclasse istanza_variabili
{flush_token_stack();conta_dichiaratore=0;}
lista_definizione_implementazioni END
| OBJC_IMPLEMENTATION nome_classe
{flush_token_stack();conta_dichiaratore=0;} definizione_istanza_metodo
END
| OBJC_IMPLEMENTATION nome_classe
{flush_token_stack();conta_dichiaratore=0;} END
| objc_imp_synt
;

objc_imp_synt
: OBJC_IMPLEMENTATION nome_classe lista_objc_synt istanza_variabili
{flush_token_stack();conta_dichiaratore=0;}
lista_definizione_implementazioni END
| OBJC_IMPLEMENTATION nome_classe ':' nome_superclasse lista_objc_synt
istanza_variabili {flush_token_stack();conta_dichiaratore=0;}
lista_definizione_implementazioni END
| OBJC_IMPLEMENTATION nome_classe lista_objc_synt
{flush_token_stack();conta_dichiaratore=0;} definizione_istanza_metodo
END
;

lista_objc_synt
: objc_synt
| lista_objc_synt objc_synt
;

objc_synt
: OBJC_SYNTH selettore_prop ';'
;
```

Questo nodo serve per la riduzione del costrutto @synthesize

```
objc_property
: OBJC_PROP '(' lista_prop ')' OBJC_CONSTANT dichiarazione
| OBJC_PROP '(' lista_prop ')' dichiarazione
;

lista_prop
: prop
| lista_prop ',' prop
;

prop
: OBJC_RETAIN
| OBJC_NONATOMIC
| OBJC_COPY
| OBJC_ASSIGN
| OBJC_READWRITE
| OBJC_READONLY
;
```

“Prop” punta direttamente alle foglie riguardanti il costrutto @property.

```
categoria_interfaccia
: OBJC_INTERFACE nome_classe '(' nome_categoria ')'
lista_dichiarazioni_interfaccia END
| OBJC_INTERFACE nome_classe '(' nome_categoria ')'
lista_protocollo_riferimento lista_dichiarazioni_interfaccia END
;

categoria_implementazione
: OBJC_IMPLEMENTATION nome_classe '(' nome_categoria ')'
lista_definizione_implementazioni END
;

dichiarazione_protocollo
: PROTOCOL nome_protocollo lista_dichiarazioni_interfaccia END
| PROTOCOL nome_protocollo lista_protocollo_riferimento
lista_dichiarazioni_interfaccia END
;

lista_dichiarazione_classe
: OBJC_ATCLASS lista_classi
;

lista_classi
: nome_classe
| nome_classe ';' {     if(controlla (p)==0)
    putsimbolo(p,"class name"); }
| lista_classi ',' nome_classe
| lista_classi ',' nome_classe ';'
;

lista_protocollo_riferimento
: '<' lista_protocolli '>'
;
```

```
lista_protocolli
: nome_protocollo
| lista_protocolli ',' nome_protocollo
;


nome_classe
: IDENTIFIER
;

nome_superclasse
: IDENTIFIER
| OBJC_CLASS_NAME
;

nome_categoria
: IDENTIFIER
;

nome_protocollo
: IDENTIFIER
;

istanza_variabili
: '{' lista_dichiarazione_struct '}'
| '{' specificazione_visibilita lista_dichiarazione_struct'}'
| '{' lista_dichiarazione_struct istanza_variabili'}'
| '{' specificazione_visibilita lista_dichiarazione_struct
istanza_variabili '}'
;

specificazione_visibilita
: PRIVATE
| PUBLIC
| PROTECTED
;

lista_dichiarazioni_interfaccia
: dichiarazione
| dichiarazione_metodo
| lista_dichiarazioni_interfaccia dichiarazione
| lista_dichiarazioni_interfaccia dichiarazione_metodo
| objc_property
| lista_dichiarazioni_interfaccia objc_property
;

dichiarazione_metodo
: dichiarazione_metodo_classe
| dichiarazione_istanza_metodo
;

dichiarazione_metodo_classe
: '+' selettore_metodo ';'
| '+' tipo_metodo selettore_metodo ';'
;

dichiarazione_istanza_metodo
```

```
: '-' selettore_metodo ';'
| '-' tipo_metodo selettore_metodo ';'
;

lista_definizione_implementazioni
: definizione_funzione
| dichiarazione
| definizione_metodo
| lista_definizione_implementazioni definizione_funzione
| lista_definizione_implementazioni dichiarazione
| lista_definizione_implementazioni definizione_metodo
;

definizione_metodo
: definizione_metodo_classe
| definizione_istanza_metodo
;

definizione_metodo_classe
: '+' selettore_metodo istruzione_composta
| '+' tipo_metodo selettore_metodo istruzione_composta
| '+' selettore_metodo lista_dichiarazioni istruzione_composta
| '+' tipo_metodo selettore_metodo lista_dichiarazioni
istruzione_composta
;

definizione_istanza_metodo
: '-' selettore_metodo {flush_token_stack();conta_dichiaratore=0;}
istruzione_composta {flush_token_stack();conta_dichiaratore=0;int
flag=0;rec = stable;
     while(rec!=NULL && match(rec->nome,"^.*.nome.funz$")==0) {
      rec=rec->prox;
     }
     flag1=0;
     if(returnflag==0) {
      if(match(rec->info,"^.*void.*$")==0) {
            sprintf(provv,"Errore, manca il return\n");
            yyerror(provv);
      }
     }
     else
      returnflag=0;

      rec=stable;

      if(stable!=NULL) {
            if(match(rec->nome,"^.*nome_funz.*$")==1)
                  flag=1;
            while(stable->prox!=NULL && flag==0) {
                  rec=stable->prox;
                  free(stable);
                  stable=rec;
                  if(match(rec->nome,"^.*nome_funz.*$")==1)
                        flag=1;
            }
      }
     dealloc_flag=0;
```

```
}
| definizione_istanza_metodo '-' selettore_metodo
{flush_token_stack();conta_dichiaratore=0;} istruzione_composta
{flush_token_stack();conta_dichiaratore=0;int flag=0;rec = stable;
     while(rec!=NULL && match(rec->nome,"^.*.nome.funz$")==0) {
      rec=rec->prox;
     }
     flag1=0;
     if(returnflag==0) {
      if(match(rec->info,"^.*void.*$")==0) {
            sprintf(provv,"Errore, manca il return\n");
            yyerror(provv);
      }
     }
     else
      returnflag=0;

      rec=stable;

      if(stable!=NULL) {
            if(match(rec->nome,"^.*nome_funz.*$")==1)
                  flag=1;
            while(stable->prox!=NULL && flag==0) {
                  rec=stable->prox;
                  free(stable);
                  stable=rec;
                  if(match(rec->nome,"^.*nome_funz.*$")==1)
                        flag=1;
            }
      }
     dealloc_flag=0;
}
| '-' tipo_metodo selettore_metodo
{flush_token_stack();conta_dichiaratore=0;} istruzione_composta
{flush_token_stack();conta_dichiaratore=0;int flag=0;rec = stable;

     while(rec!=NULL && match(rec->nome,"^.*.nome.funz$")==0) {
      rec=rec->prox;
     }
     flag1=0;
     if(returnflag==0) {
      if(match(rec->info,"^.*void.*$")==0) {
            sprintf(provv,"Errore, manca il return\n");
            yyerror(provv);
      }
     }
     else
      returnflag=0;

      rec=stable;

      if(stable!=NULL) {
            if(match(rec->nome,"^.*nome_funz.*$")==1)
                  flag=1;
            while(stable->prox!=NULL && flag==0) {
                  rec=stable->prox;
                  free(stable);
```

```
                stable=rec;
                if(match(rec->nome,"^.*nome_funz.*$")==1)
                      flag=1;
            }
      }
     dealloc_flag=0;
}
| definizione_istanza_metodo '-' tipo_metodo selettore_metodo
{flush_token_stack();conta_dichiaratore=0;} istruzione_composta
{flush_token_stack();conta_dichiaratore=0;int flag=0;rec = stable;
     while(rec!=NULL && match(rec->nome,"^.*.nome.funz$")==0) {
      rec=rec->prox;
     }
     flag1=0;
     if(returnflag==0) {
      if(match(rec->info,"^.*void.*$")==0) {
            sprintf(provv,"Errore, manca il return\n");
            yyerror(provv);
      }
     }
     else
      returnflag=0;

      rec=stable;

      if(stable!=NULL) {
            if(match(rec->nome,"^.*nome_funz.*$")==1)
                  flag=1;
            while(stable->prox!=NULL && flag==0) {
                  rec=stable->prox;
                  free(stable);
                  stable=rec;
                  if(match(rec->nome,"^.*nome_funz.*$")==1)
                        flag=1;
            }
      }
     dealloc_flag=0;
}
| '-' selettore_metodo lista_dichiarazioni
{flush_token_stack();conta_dichiaratore=0;} istruzione_composta
{flush_token_stack();conta_dichiaratore=0;int flag=0;rec = stable;
     while(rec!=NULL && match(rec->nome,"^.*.nome.funz$")==0) {
      rec=rec->prox;
     }
     flag1=0;
     if(returnflag==0) {
      if(match(rec->info,"^.*void.*$")==0) {
            sprintf(provv,"Errore, manca il return\n");
            yyerror(provv);
      }
     }
     else
      returnflag=0;

      rec=stable;

      if(stable!=NULL) {
```

```
            if(match(rec->nome,"^.*nome_funz.*$")==1)
                  flag=1;
            while(stable->prox!=NULL && flag==0) {
                  rec=stable->prox;
                  free(stable);
                  stable=rec;
                  if(match(rec->nome,"^.*nome_funz.*$")==1)
                        flag=1;
            }
      }
     dealloc_flag=0;
}
| definizione_istanza_metodo '-' selettore_metodo lista_dichiarazioni
{flush_token_stack();conta_dichiaratore=0;} istruzione_composta
{flush_token_stack();conta_dichiaratore=0;int flag=0;rec = stable;
     while(rec!=NULL && match(rec->nome,"^.*.nome.funz$")==0) {
      rec=rec->prox;
     }
     flag1=0;
     if(returnflag==0) {
      if(match(rec->info,"^.*void.*$")==0) {
            sprintf(provv,"Errore, manca il return\n");
            yyerror(provv);
      }
     }
     else
      returnflag=0;

      rec=stable;

      if(stable!=NULL) {
            if(match(rec->nome,"^.*nome_funz.*$")==1)
                  flag=1;
            while(stable->prox!=NULL && flag==0) {
                  rec=stable->prox;
                  free(stable);
                  stable=rec;
                  if(match(rec->nome,"^.*nome_funz.*$")==1)
                        flag=1;
            }
      }
     dealloc_flag=0;
}
| '-' tipo_metodo selettore_metodo lista_dichiarazioni
{flush_token_stack();conta_dichiaratore=0;} istruzione_composta
{flush_token_stack();conta_dichiaratore=0;int flag=0;rec = stable;
     while(rec!=NULL && match(rec->nome,"^.*.nome.funz$")==0) {
      rec=rec->prox;
     }
     flag1=0;
     if(returnflag==0) {
      if(match(rec->info,"^.*void.*$")==0) {
            sprintf(provv,"Errore, manca il return\n");
            yyerror(provv);
      }
     }
     else
```

```
      returnflag=0;

      rec=stable;

      if(stable!=NULL) {
            if(match(rec->nome,"^.*nome_funz.*$")==1)
                  flag=1;
            while(stable->prox!=NULL && flag==0) {
                  rec=stable->prox;
                  free(stable);
                  stable=rec;
                  if(match(rec->nome,"^.*nome_funz.*$")==1)
                        flag=1;
            }
      }
     dealloc_flag=0;
}
| definizione_istanza_metodo '-' tipo_metodo selettore_metodo
lista_dichiarazioni {flush_token_stack();conta_dichiaratore=0;}
istruzione_composta{flush_token_stack();conta_dichiaratore=0;int
flag=0;rec = stable;
     while(rec!=NULL && match(rec->nome,"^.*.nome.funz$")==0) {
      rec=rec->prox;
     }
     flag1=0;
     if(returnflag==0) {
      if(match(rec->info,"^.*void.*$")==0) {
            sprintf(provv,"Errore, manca il return\n");
            yyerror(provv);
      }
     }
     else
      returnflag=0;

      rec=stable;

      if(stable!=NULL) {
            if(match(rec->nome,"^.*nome_funz.*$")==1)
                  flag=1;
            while(stable->prox!=NULL && flag==0) {
                  rec=stable->prox;
                  free(stable);
                  stable=rec;
                  if(match(rec->nome,"^.*nome_funz.*$")==1)
                        flag=1;
            }
      }
     dealloc_flag=0;
}
;
```

Questo nodo è l'omologo di "definizione_funzione" riferito ai metodi di istanza dell'objc, pertanto, valgono le considerazioni fatte in precedenza.

```
selettore_metodo
: selettore_unario
| selettore_parolechiave
| selettore_parolechiave ',' ELLIPSIS
| selettore_parolechiave ',' lista_tipo_parametri
;

selettore_unario
: selettore {
     struct token_stack *s;
     if(dealloc_flag==0){
      strcpy(provv,p);strcat(provv,"_nome_funz");
      if(flag_funzione_puntatore == 1) {
            strcat(provv,"*");
            flag_funzione_puntatore = 0;
      }
      s=ts_head;
      while(s->prox!=NULL)
            s=s->prox;
      controlla_aggiungi(provv,s->toke);
     }else
     {
      if(match(ts_head->toke,"^.*void.*$")==1)
            controlla_aggiungi("dealloc_nome_funz",ts_head->toke);
      else
      {
            sprintf(provv,"\nErrore, la funzione dealloc prevede void come
tipo di ritorno, invece e presente %s, errore alla riga %d\n",ts_head-
>toke,numeroriga);
            yyerror(provv);
      }
      dealloc_flag=0;

     }
}
;

selettore_parolechiave
: dichiaratore_parolechiave
| selettore_parolechiave dichiaratore_parolechiave
;

dichiaratore_parolechiave
: ':' selettore_prop
| ':' tipo_metodo  selettore_prop { controlla_aggiungi(ts_head-
>tname,ts_head->prox->tname);}
| selettore ':' {strcpy(provv,p);strcat(provv,"_nome_funz");
     struct token_stack *s=ts_head;
     while(s->prox!=NULL)
      s=s->prox;
      controlla_aggiungi(provv,s->toke);
         } selettore_prop
| selettore ':' {strcpy(provv,p);strcat(provv,"_nome_funz");
     struct token_stack *s=ts_head;
     s=ts_head;
     while(s->prox!=NULL)
      s=s->prox;
```

```
      controlla_aggiungi(provv,s->toke);
      } tipo_metodo selettore_prop { controlla_aggiungi(ts_head-
>tname,ts_head->prox->tname);}
;


selettore
: IDENTIFIER
| OBJC_INSTANCE_METHOD
| OBJC_DEALLOC {dealloc_flag=1;}
| OBJC_PROTOCOL_METHOD
| OBJC_CLASS_NAME
;

selettore_prop
: IDENTIFIER
| OBJC_INSTANCE_PROPERTY {
     struct token_stack *i=ts_head;
     for(i=ts_head;i->prox!=NULL;i=i->prox){}
     controlla_aggiungi(p,i->tname);

}
| OBJC_INSTANCE_METHOD {
     struct token_stack *i=ts_head;
     for(i=ts_head;i->prox!=NULL;i=i->prox){}
     controlla_aggiungi(p,i->tname);

}
| OBJC_PROTOCOL_METHOD{
     struct token_stack *i=ts_head;
     for(i=ts_head;i->prox!=NULL;i=i->prox){}
     controlla_aggiungi(p,i->tname);

}
| STRING_LITERAL
| OBJC_CLASS_METHOD{
     struct token_stack *i=ts_head;
     for(i=ts_head;i->prox!=NULL;i=i->prox){}
     controlla_aggiungi(p,i->tname);

}
;

selettore_prop2
: OBJC_INSTANCE_PROPERTY
| OBJC_INSTANCE_METHOD
| OBJC_PROTOCOL_METHOD
| OBJC_CLASS_METHOD
;
```

Questi selettori sono nodi che puntano direttamente alle foglie dell'indicatore sintagmatico ad albero. Vengono richiamati da differenti non terminali e sono in parte ridondanti per evitare errori

di tipo reduce/reduce, ovvero situazioni di forte ambiguità non risolvibili con un lookahead di un token.

```
tipo_metodo
: '(' nome_tipo ')'
;
```

La sezione *epilogo*:

```
int main(int argc, char* argv[])
{

    extern FILE *yyin;
    extern FILE *yyout;
    char path[256];

    strcpy(path,argv[1]);
    struct token_stack *p=NULL;
    //extern int yydebug;
    //yydebug=1;
    carica_stable(path);

    fflush(stdin);
    fflush(stdout);
    yyout=fopen( "output","w");

    if((yyin = fopen( argv[2], "r" ))!=NULL)
    {

        yyparse();
        printf("Computazione eseguita");
        return 0;

    } else
    {
        yyin=stdin;
        yyparse();

        printf("Computazione eseguita");
        return 0;
    }


}

int yyerror ( char *s)
{
    errori++;
    printf("FAIL:%s ", s);
    return 1;

}
```

Il main qui dichiarato carica la tabella dei simboli e provvede a redirigere lo stream di input se ne viene specificato uno come argomento della chiamata a blocK. Dopo aver aperto anche uno stream di output chiama la funzione yyparse() che fa incominciare la computazione.

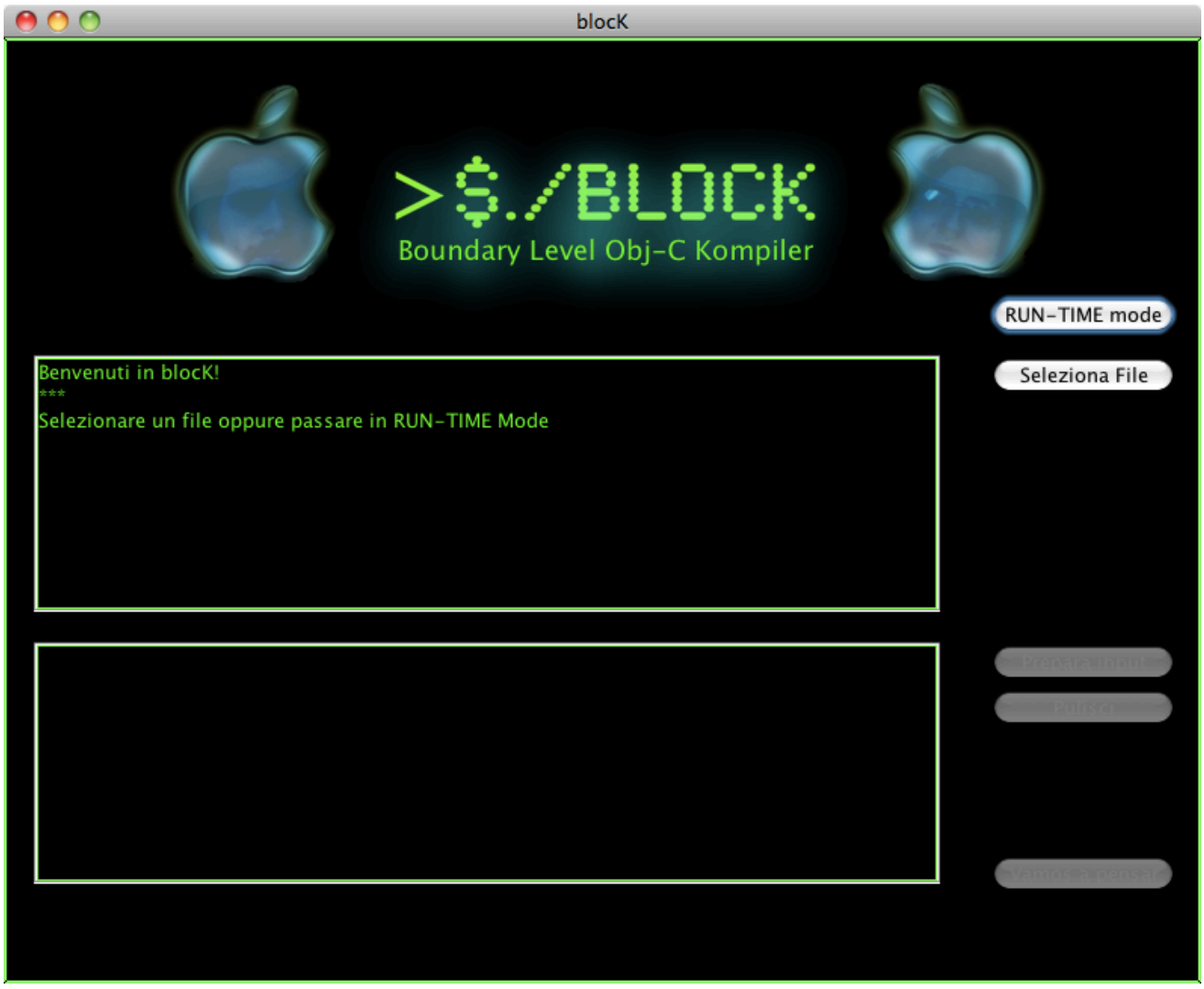
La ridefinizione della funzione yyerror() è necessaria per migliorare la stampa a video degli errori comprendendo anche quelli definiti dal programmatore già specificati in precedenza.

## *L'interfaccia utente*

JBlock deve essere eseguito tramite il comando:

*java –jar /path_del_jar /path_di_block /path_elenco_stable*

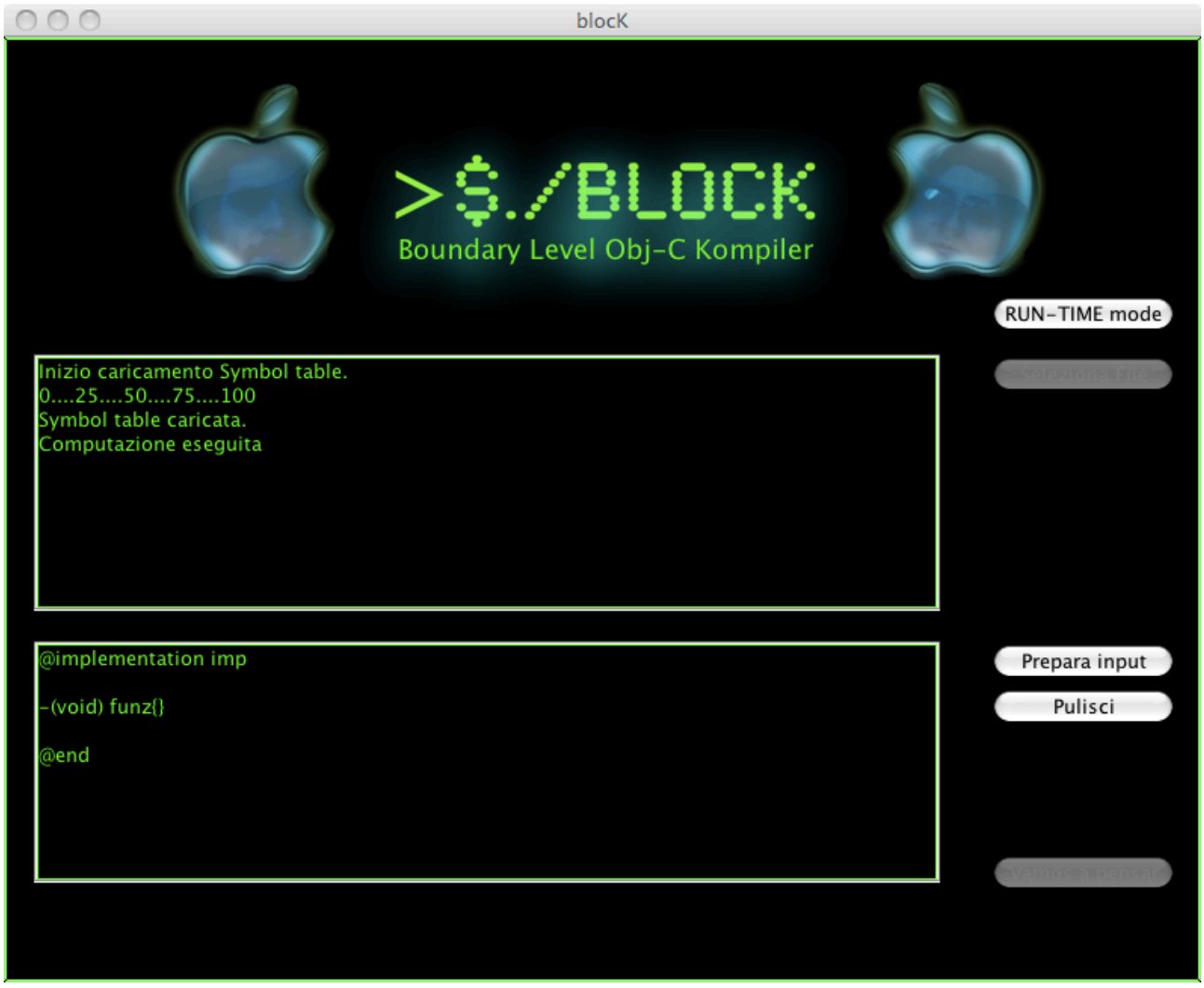
e si presenta con la seguente schermata:




La GUI è stata implementata in linguaggio Java e si compone di due aree di testo, una di input ed una di output.

Il programma parte in modalità file, ovvero prevede che l'utente selezioni un file mediante l'apposito pulsante che attiverà un file Chooser. Se la scelta del file è andata a buon fine, JBlock attiverà il pulsante di inizio computazione.

Attivando la modalità runtime viene disabilitato il pulsante di ricerca del file e viene attivata l'area di testo di input. A questo punto l'utente può specificare del codice e prepararlo all'analisi. La pressione del pulsante "prepara input" attiva il pulsante di inizio computazione. È inoltre disponibile un ulteriore pulsante per pulire il terminale di input. L'utente può in ogni momento cambiare modalità di utilizzo.



## *Casi di test*

blocK è stato testato con due applicazioni per iPhone: "Ciao" e "Convenevoli".

La prima è una specie di "Hello World" che, oltre a stampare a video il benvenuto, si serve delle routine dell'accelerometro per far sì che l'etichetta ruoti al ruotare del telefono; la seconda, invece, risponde al saluto una volta che l'utente ha inserito il proprio nome tramite l'apposito campo ed ha confermato la scelta con l'apposito pulsante.


Screenshot di "ciao"

## File di input: main.m

```
//
//  main.m
//  ciao
//
//  Created by vito on 20/03/10.
//  Copyright __MyCompanyName__ 2010. All rights reserved.
//

#import <UIKit/UIKit.h>

int main(int argc, char *argv[]) {

    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    int retVal = UIApplicationMain(argc, argv, nil, nil);
    [pool release];
    return retVal;
}
```

---

## blocK restituisce:

```
Inizio caricamento Symbol table.
0....25....50....75....100
Symbol table caricata.

  main.m
  ciao

  Created by vito on 20/03/10.
  Copyright __MyCompanyName__ 2010. All rights reserved.

 <UIKit/UIKit.h>
Computazione eseguita
```

## File di input: ciaoAppDelegate.h

```
//
//  ciaoAppDelegate.h
//  ciao
//
//  Created by vito on 20/03/10.
//  Copyright __MyCompanyName__ 2010. All rights reserved.
//

#import <UIKit/UIKit.h>

@class ciaoViewController;

@interface ciaoAppDelegate : NSObject <UIApplicationDelegate> {
    UIWindow *window;
    ciaoViewController *viewController;
}

@property (nonatomic, retain) IBOutlet UIWindow *window;
@property (nonatomic, retain) IBOutlet ciaoViewController
*viewController;

@end
```

---

## blocK restituisce:

```
Inizio caricamento Symbol table.

0....25....50....75....100

Symbol table caricata.


  ciaoAppDelegate.h

  ciao


  Created by vito on 20/03/10.

  Copyright __MyCompanyName__ 2010. All rights reserved.


 <UIKit/UIKit.h>

Computazione eseguita
```

## File di input: ciaoAppDelegate.m

```
//
//  ciaoAppDelegate.m
//  ciao
//
//  Created by vito on 20/03/10.
//  Copyright __MyCompanyName__ 2010. All rights reserved.
//

#import "ciaoAppDelegate.h"
#import "ciaoViewController.h"

@implementation ciaoAppDelegate

@synthesize window;
@synthesize viewController;


- (void)applicationDidFinishLaunching:(UIApplication *)application {

    // Override point for customization after app launch
    [window addSubview:viewController.view];
    [window makeKeyAndVisible];
}


- (void)dealloc {
    [viewController release];
    [window release];
    [super dealloc];
}


@end
```

---

## blocK restituisce:

```
Inizio caricamento Symbol table.
0....25....50....75....100
Symbol table caricata.

  ciaoAppDelegate.m
  ciao

  Created by vito on 20/03/10.
  Copyright __MyCompanyName__ 2010. All rights reserved.

 "ciaoAppDelegate.h"
 "ciaoViewController.h"
 Override point for customization after app launch
Computazione eseguita
```

## File di input: ciaoViewController.h

```
//
//  ciaoViewController.h
//  ciao
//
//  Created by vito on 20/03/10.
//  Copyright __MyCompanyName__ 2010. All rights reserved.
//

#import <UIKit/UIKit.h>

@interface ciaoViewController : UIViewController {

}

@end
```

---

## blocK restituisce:

```
Inizio caricamento Symbol table.
0....25....50....75....100
Symbol table caricata.

  ciaoViewController.h
  ciao

  Created by vito on 20/03/10.
  Copyright __MyCompanyName__ 2010. All rights reserved.

 <UIKit/UIKit.h>
Computazione eseguita
```

## File di input: ciaoViewController.m

```
//
//  ciaoViewController.m
//  ciao
//
//  Created by vito on 20/03/10.
//  Copyright __MyCompanyName__ 2010. All rights reserved.
//

#import "ciaoViewController.h"

@implementation ciaoViewController



/*
// The designated initializer. Override to perform setup that is
required before the view is loaded.
- (id)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle
*)nibBundleOrNil {
    if (self = [super initWithNibName:nibNameOrNil
bundle:nibBundleOrNil]) {
        // Custom initialization
    }
    return self;
}
*/

/*
// Implement loadView to create a view hierarchy programmatically,
without using a nib.
- (void)loadView {
}
*/


/*
// Implement viewDidLoad to do additional setup after loading the view,
typically from a nib.
- (void)viewDidLoad {
    [super viewDidLoad];
}
*/



// Override to allow orientations other than the default portrait
orientation.
-
(BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)int
erfaceOrientation {
    // Return YES for supported orientations
    return YES;
}
```

```
- (void)didReceiveMemoryWarning {
    // Releases the view if it doesn't have a superview.
    [super didReceiveMemoryWarning];

    // Release any cached data, images, etc that aren't in use.
}

- (void)viewDidUnload {
    // Release any retained subviews of the main view.
    // e.g. self.myOutlet = nil;
}


- (void)dealloc {
    [super dealloc];
}

@end
```

---

## blocK restituisce:

```
Inizio caricamento Symbol table.
0....25....50....75....100
Symbol table caricata.

  ciaoViewController.m
  ciao

  Created by vito on 20/03/10.
  Copyright __MyCompanyName__ 2010. All rights reserved.

 "ciaoViewController.h"

// The designated initializer. Override to perform setup that is
required before the view is loaded.
- (id)initWithNibName:(NSString )nibNameOrNil bundle:(NSBundle
)nibBundleOrNil {
    if (self = [super initWithNibName:nibNameOrNil
bundle:nibBundleOrNil]) {
        // Custom initialization
    }
    return self;
}
/
// Implement loadView to create a view hierarchy programmatically,
without using a nib.
- (void)loadView {
}
/
// Implement viewDidLoad to do additional setup after loading the view,
typically from a nib.
- (void)viewDidLoad {
    [super viewDidLoad];
```

```
}
/ Override to allow orientations other than the default portrait
orientation.
 Return YES for supported orientations
 Releases the view if it doesn't have a superview.
 Release any cached data, images, etc that aren't in use.
 Release any retained subviews of the main view.
 e.g. self.myOutlet = nil;
Computazione eseguita
```


Screenshot di “convenevoli”

## File di input: main.m

```
//
//  main.m
//  Convenevoli
//
//  Created by MacShark on 07/04/10.
//  Copyright __MyCompanyName__ 2010. All rights reserved.
//

#import <UIKit/UIKit.h>

int main(int argc, char *argv[]) {



    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    int retVal = UIApplicationMain(argc, argv, nil, nil);
    [pool release];
    return retVal;
}
```

---

## blocK restituisce:

```
Inizio caricamento Symbol table.
0....25....50....75....100
Symbol table caricata.

  main.m
  Convenevoli

  Created by MacShark on 07/04/10.
  Copyright __MyCompanyName__ 2010. All rights reserved.

 <UIKit/UIKit.h>
Computazione eseguita
```

## File di input: convenevoliAppDelegate.h

```
//
//  ConvenevoliAppDelegate.h
//  Convenevoli
//
//  Created by MacShark on 07/04/10.
//  Copyright __MyCompanyName__ 2010. All rights reserved.
//

#import <UIKit/UIKit.h>

@class ConvenevoliViewController;

@interface ConvenevoliAppDelegate : NSObject <UIApplicationDelegate> {
    UIWindow *window;
    ConvenevoliViewController *viewController;
}

@property (nonatomic, retain) IBOutlet UIWindow *window;
@property (nonatomic, retain) IBOutlet ConvenevoliViewController
*viewController;

@end
```

---

## blocK restituisce:

```
Inizio caricamento Symbol table.
0....25....50....75....100
Symbol table caricata.

  ConvenevoliAppDelegate.h
  Convenevoli

  Created by MacShark on 07/04/10.
  Copyright __MyCompanyName__ 2010. All rights reserved.

 <UIKit/UIKit.h>
Computazione eseguita
```

## File di input: convenevoliAppDelegate.m

```
//
//  ConvenevoliAppDelegate.m
//  Convenevoli
//
//  Created by MacShark on 07/04/10.
//  Copyright __MyCompanyName__ 2010. All rights reserved.
//

#import "ConvenevoliAppDelegate.h"
#import "ConvenevoliViewController.h"

@implementation ConvenevoliAppDelegate

@synthesize window;
@synthesize viewController;


- (void)applicationDidFinishLaunching:(UIApplication *)application {

    // Override point for customization after app launch
    [window addSubview:viewController.view];
    [window makeKeyAndVisible];
}

- (void)dealloc {
    [viewController release];
    [window release];
    [super dealloc];
}


@end
```

---

## blocK restituisce:

```
Inizio caricamento Symbol table.
0....25....50....75....100
Symbol table caricata.

  ConvenevoliAppDelegate.m
  Convenevoli

  Created by MacShark on 07/04/10.
  Copyright __MyCompanyName__ 2010. All rights reserved.

 "ConvenevoliAppDelegate.h"
 "ConvenevoliViewController.h"
 Override point for customization after app launch
Computazione eseguita
```

## File di input: convenevoliViewController.h

```
//
//  ConvenevoliViewController.h
//  Convenevoli
//
//  Created by MacShark on 07/04/10.
//  Copyright __MyCompanyName__ 2010. All rights reserved.
//

#import <UIKit/UIKit.h>

@interface ConvenevoliViewController : UIViewController {

    IBOutlet UILabel *helloLabel;
    IBOutlet UITextField *nameField;
    IBOutlet UILabel *salveLabel;

}

- (IBAction) sayHello: (id) sender;

@end
```

---

## blocK restituisce:

```
Inizio caricamento Symbol table.
0....25....50....75....100
Symbol table caricata.

  ConvenevoliViewController.h
  Convenevoli

  Created by MacShark on 07/04/10.
  Copyright __MyCompanyName__ 2010. All rights reserved.

 <UIKit/UIKit.h>
Computazione eseguita
```

## File di input: convenevoliViewController.m

```
//
//  ConvenevoliViewController.m
//  Convenevoli
//
//  Created by MacShark on 07/04/10.
//  Copyright __MyCompanyName__ 2010. All rights reserved.
//

#import "ConvenevoliViewController.h"

@implementation ConvenevoliViewController



/*
// The designated initializer. Override to perform setup that is
required before the view is loaded.
- (id)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle
*)nibBundleOrNil {
    if (self = [super initWithNibName:nibNameOrNil
bundle:nibBundleOrNil]) {
        // Custom initialization
    }
    return self;
}
*/

/*
// Implement loadView to create a view hierarchy programmatically,
without using a nib.
- (void)loadView {
}
*/



// Implement viewDidLoad to do additional setup after loading the view,
typically from a nib.
- (void)viewDidLoad {
    [super viewDidLoad];
}




// Override to allow orientations other than the default portrait
orientation.
-
(BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)int
erfaceOrientation {
    // Return YES for supported orientations C'era un BOOL nel ritorno
    return (YES);
}
```

```
- (void)didReceiveMemoryWarning {
    // Releases the view if it doesn't have a superview.
    [super didReceiveMemoryWarning];

    // Release any cached data, images, etc that aren't in use.
}

- (void)viewDidUnload {
    // Release any retained subviews of the main view.
    // e.g. self.myOutlet = nil
}


- (void)dealloc {
    [super dealloc];
}

- (void) sayHello:(id)sender{

    NSString *userName=nameField.text;
    NSString *helloMessage=[[NSString alloc] initWithFormat:@"Salve %@",
userName];
    helloLabel.text=helloMessage;
    [helloMessage release];
    nameField.text=NULL;
}

@end
```

---

## blocK restituisce:

```
Inizio caricamento Symbol table.
0....25....50....75....100
Symbol table caricata.

  ConvenevoliViewController.m
  Convenevoli

  Created by MacShark on 07/04/10.
  Copyright __MyCompanyName__ 2010. All rights reserved.

 "ConvenevoliViewController.h"

// The designated initializer. Override to perform setup that is
required before the view is loaded.
- (id)initWithNibName:(NSString )nibNameOrNil bundle:(NSBundle
)nibBundleOrNil {
    if (self = [super initWithNibName:nibNameOrNil
bundle:nibBundleOrNil]) {
        // Custom initialization
    }
    return self;
```

```
}
/
// Implement loadView to create a view hierarchy programmatically,
without using a nib.
- (void)loadView {
}
/ Implement viewDidLoad to do additional setup after loading the view,
typically from a nib.
 Override to allow orientations other than the default portrait
orientation.
 Return YES for supported orientations C'era un BOOL nel ritorno
 Releases the view if it doesn't have a superview.
 Release any cached data, images, etc that aren't in use.
 Release any retained subviews of the main view.
 e.g. self.myOutlet = nil
Computazione eseguita
```

## File di input: equazione_secondo_grado.c

```
#include <stdio.h>
#include <math.h>


int main()
{
     //dichiarazione variabili
     float a, b, c, delta, x1, x2;

     //inserimento dei coefficienti a, b, c di un'equazione di secondo
grado
     printf("Inserisci i coefficienti a, b, c: ");
     scanf("%f %f %f", &a, &b, &c);

     //calcolo del delta
     delta = b*b - 4*a*c;

     //calcolo delle radici
     if(delta >= 0) {
      x1 = (-b + sqrt(delta)) / (2*a);
      x2 = (-b - sqrt(delta)) / (2*a);

      printf("\n\nLe radici dell'equazione %.2fx^2 + %.2fx + %.2f sono:
", a, b, c);
      printf("\n\t x1 = %.2f \n\t x2 = %.2f", x1, x2);
     } else {
      printf("\n\nL'equazione %.2fx^2 + %.2fx + %.2f ha radici
immaginarie.", a, b, c);
     }


     printf("\n\nPremi INVIO per uscire.");
     //in questo caso sono necessarie due chiamate a getchar()
     getchar(); getchar();
     return 0;
}
```

---

## blocK restituisce:

```
Inizio caricamento Symbol table.
0....25....50....75....100
Symbol table caricata.
 <stdio.h>
 <math.h>
dichiarazione variabili
inserimento dei coefficienti a, b, c di un'equazione di secondo grado
calcolo del delta
calcolo delle radici
in questo caso sono necessarie due chiamate a getchar()
Computazione eseguita
```

## File di input: fattoriale.c

```
#include <stdio.h>
#include <stdlib.h>

int fattoriale(unsigned long);

int main()
{
     //dichiarazione di variabili
     unsigned int num;
     char risp;

     do {
      do {
            //pulisce lo schermo
            system("cls");

            printf("Inserisci un numero minore di 13 di cui vuoi calcolare
il fattoriale: ");
            scanf("%u", &num);
      } while(num < 0 || num >= 13);

      printf("\nIl fattoriale \212: %lu", fattoriale(num));

      printf("\n\nVuoi calcolare un altro fattoriale? [s/n] ");
      scanf("\n%c", &risp);
      //scarta il carattere di ritorno a capo
      getchar();
     } while(risp == 's' || risp == 'S');


     printf("\n\nPremi INVIO per uscire.");
     getchar();
     return 0;
}


int fattoriale(unsigned long n)
{
     if(!n || n == 1) {
      return 1;
     } else {
      return n * fattoriale(n - 1);
     }
}
```

---

## blocK restituisce:

```
Inizio caricamento Symbol table.
0....25....50....75....100
Symbol table caricata.
 <stdio.h>
 <stdlib.h>
```

```
dichiarazione di variabili
pulisce lo schermo
scarta il carattere di ritorno a capo
Computazione eseguita
```

## File di input: ordinamento_bubble_sort.c

```
#include <stdio.h>


int main()
{
     //dichiarazione di variabili
     int i, j, temp;
     int vettore[6];

     //caricamento del vettore
     for(i = 0; i < 6; i++) {
      printf("Inserisci il %d\370 valore: ", i+1);
      scanf("%d", &vettore[i]);
     }

     //ordinamento vettore
     for(i = 6; i > 1; i--) {
      for(j = 0; j < i - 1; j++) {
            if(vettore[j] > vettore[j+1]) {
                  temp          = vettore[j];
                  vettore[j]    = vettore[j+1];
                  vettore[j+1] = temp;
            }
      }
     }

     //visualizzazione vettore ordinato
     printf("Il vettore ordinato \212: ");
     for(i = 0; i < 6; i++) {
      printf("%d ", vettore[i]);
     }

     printf("\n\nPremi INVIO per uscire.");
     //sono necessarie due chiamate a getchar in questo caso
     getchar();getchar();
     return 0;
}
```

---

## blocK restituisce:

```
Inizio caricamento Symbol table.
0....25....50....75....100
Symbol table caricata.
 <stdio.h>
dichiarazione di variabili
caricamento del vettore
ordinamento vettore
visualizzazione vettore ordinato
sono necessarie due chiamate a getchar in questo caso
Computazione eseguita
```

# *Gestione degli errori*

## File di input: errori_inizializzazioni.c

```
int restituisco_intero()
{
     int a=5;
     return a;
}

int* restituisco_p_intero()
{
     return NULL;
}

void main()
{
     int a=1;
     int *b=NULL;
     int c=a;
     float d=a;
     int *e=b;
     float *f=b;
     int g=restituisco_intero();
     char h=restituisco_intero();
     int *i=restituisco_intero();
     int *l=restituisco_p_intero();
     char m=restituisco_p_intero();
     double n=restituisco_p_intero();
     double *o=restituisco_p_intero();

}
```

---

## blocK restituisce:

```
Inizio caricamento Symbol table.
0....25....50....75....100
Symbol table caricata.
FAIL:
Errore tipi variabile a dx e sx dell'inizializzazione, probabilmente
sarebbe opportuno controllare la riga 17
 FAIL:
Errore tipi puntatori a dx e sx dell'inizializzazione, probabilmente
sarebbe opportuno controllare la riga 19
 FAIL:
Errore tipi variabile a dx e sx dell'inizializzazione, probabilmente
sarebbe opportuno controllare la riga 21
 FAIL:
 Si sta inizializzando una variabile puntatore con il tipo di ritorno
della funzione restituisco_intero che non lo è, errore alla riga 22
 FAIL:
Si sta inizializzando una variabile con il tipo di ritorno della
funzione restituisco_p_intero che è un puntatore, errore alla riga 24
```

```
 FAIL:
Si sta inizializzando una variabile con il tipo di ritorno della
funzione restituisco_p_intero che è un puntatore, errore alla riga 25
 FAIL:
Errore tipi puntatori a dx e sx dell'inizializzazione, probabilmente
sarebbe opportuno controllare la riga 26
 Computazione eseguita
```

## File di input: errori_assegnamento.c

```
int restituisco_intero()
{
     int a=5;
     return a;
}

int* restituisco_p_intero()
{
     return NULL;
}

void main()
{
     int a,b,c;
     float d;
     float *e;
     int *i;
     a=1;
     b=2;
     c=a*b;
     d=1.5;
     e=NULL;
     c=d;
     e=b;
     i=restituisco_p_intero();
     e=restituisco_p_intero();
     a=restituisco_p_intero();
     c=restituisco_intero();
     d=restituisco_intero();
     e=restituisco_intero();
     i=restituisco_intero();
}
```

---

## blocK restituisce:

```
Inizio caricamento Symbol table.
0....25....50....75....100
Symbol table caricata.
FAIL:
Errore tipi variabile a dx e sx dell'assegnamento, probabilmente sarebbe
opportuno controllare la riga 23
```

```
 FAIL:
Variabile b non compatibile nell'assegnamento, errore alla riga 24
 FAIL:
Errore tipi puntatori a dx e sx dell'inizializzazione, probabilmente
sarebbe opportuno controllare la riga 26
 FAIL:
Si sta assegnando alla variabile a il tipo di ritorno della funzione
restituisco_p_intero che è un puntatore, errore alla riga 27
 FAIL:
Errore tipi variabile a dx e sx dell'assegnamento, probabilmente sarebbe
opportuno controllare la riga 29
 FAIL:
 Alla variabile puntatore e si sta assegnando il tipo di ritorno della
funzione restituisco_intero che non lo è, errore alla riga 30
 FAIL:
 Alla variabile puntatore i si sta assegnando il tipo di ritorno della
funzione restituisco_intero che non lo è, errore alla riga 31
 Computazione eseguita
```

## File di input: errore_end.m

```
#import <UIKit/UIKit.h>

@implementation implementazione

-(int)funzione1:(int)a{
     int b=2;
     if(a==b)
      b=2*a;
     else
      b+=a;
     return b;
}
```

---

## blocK restituisce:

```
Inizio caricamento Symbol table.
0....25....50....75....100
Symbol table caricata.
 <UIKit/UIKit.h>
FAIL:syntax error, unexpected $end, expecting END or '-' .
Computazione eseguita
```

## File di input: errore_inizializzazione_assegnamento.m

```
#import <UIKit/UIKit.h>

@implementation implementazione

-(void)metodo {
    NSString *str = NULL;
    NSString *str1 = [NSString alloc];
    NSString *str2 = [UIApplication alloc];
    UIApplication *app = [IBOutlet alloc];
    NSString *s = str1;
    NSString *s1 = 12;

    str1=NULL;
    str=nil;
    str = str1;
    intero=NIL;
    intero=12;
}

@end
```

---

## blocK restituisce:

```
Inizio caricamento Symbol table.
0....25....50....75....100
Symbol table caricata.
 <UIKit/UIKit.h>
FAIL:
Errore allocazione NSString = UIApplication, errore alla riga 8
 FAIL:
Errore allocazione UIApplication = IBOutlet, errore alla riga 9
 FAIL:
Variabile s1* non compatibile nell'inizializzazione, errore alla riga 11
 FAIL:
Variabile NIL non presente in ST, errore alla riga 16
 FAIL:
Variabile intero non presente in ST, errore alla riga 17
 Computazione eseguita
```

## File di input: errore_funzione_non_presente.c

```
float funzione() {
     float x;
     return x;
}

float funzione3(float a, char b) {
     float x = a+3;
     return x;
}

int main() {
     float x;
     char v;

     x=funzione();
     x=funzione2();
     x=funzione3(x,v);
     return 0;
}
```

---

## blocK restituisce:

```
Inizio caricamento Symbol table.
0....25....50....75....100
Symbol table caricata.
FAIL:
Variabile funzione2 non presente in ST, errore alla riga 17
 Computazione eseguita
```

## File di input: errore_return.c

```
int f1(){
    return;
}

void f2(){
    return;
}

float f3(int a) {
    return a;
}

int f4() {
    return 5;
}
```

---

## blocK restituisce:

```
Inizio caricamento Symbol table.
0....25....50....75....100
Symbol table caricata.
FAIL:Errore incompatibilità tipo la funzione prevede int e invece non
restituisce alcun valore, errore alla riga 2
FAIL:Errore, la funzione restituisce a, invece dovrebbe restituire
float, errore alla riga 10
 Computazione eseguita
```

## File di input: errore_return.m

```
@implementation implementazione

-(int)f1 {
    return;
}

-(void)f2 {
    return;
}

-(float) f3: (int) a {
    return a;
}

-(int) f4 {
    return 5;
}

-(NSString) f5 {
    NSString *str = [NSString alloc];

    return str;
}

-(NSString*) f6 {
    NSString *str = [NSString alloc];

    return str;
}

-(BOOL) f7 {
    return 5;
}

-(BOOL) f8 {
    return YES;
}

@end
```

---

## blocK restituisce:

```
Inizio caricamento Symbol table.
0....25....50....75....100
Symbol table caricata.
FAIL:Errore incompatibilità tipo la funzione prevede int e invece non
restituisce alcun valore, errore alla riga 4
 FAIL:Errore, la funzione restituisce a, invece dovrebbe restituire
float, errore alla riga 12
 FAIL:Errore, la funzione restituisce un puntatore ad objc_class_name,
invece dovrebbe restituire un objc_class_name, errore alla riga 22
```

```
 FAIL:Errore incompatibilità tipo la funzione dovrebbe restituire un
booleano, errore alla riga 32
 Computazione eseguita
```

## File di input: errore_gestione_allocazione.m

```
@implementation implementazione

-(NSString*) f1 {
    NSString *g=[NSString alloc];
    NSString tr=[NSString alloc];
    NSString *c=[UIApplication alloc];
    NSString y=[UIApplication alloc];

    return y;
}

@end
```

---

## blocK restituisce:

```
Inizio caricamento Symbol table.
0....25....50....75....100
Symbol table caricata.
FAIL:
Errore allocazione, si vuole allocare NSString che non è un puntatore,
errore alla riga 5
 FAIL:
Errore allocazione NSString = UIApplication, errore alla riga 6
 FAIL:
Errore allocazione, si vuole allocare UIApplication che non è un
puntatore, errore alla riga 7
 FAIL:Errore, la funzione restituisce y, invece dovrebbe restituire un
puntatore ad objc_class_name, errore alla riga 9
 Computazione eseguita
```

## *Limitazioni*

È indubbio il fatto che una coppia di analizzatori realizzati in questa maniera presenti delle limitazioni dovute, ad esempio, alla mancanza della gestione del linking tra i files, il che implica l'impossibilità di riconoscere e gestire svariati tipi di errore.
Un progetto realizzato in Objective C, come in qualsiasi altro linguaggio di programmazione, prevede che si realizzino delle librerie ad hoc o che se ne includano alcune standard, questo implicherebbe il caricamento di una adeguata tabella dei simboli realizzata diversamente rispetto alla lista non ordinata che è stata qui implementata a scopo puramente didattico.
Disporre di una base di dati che gestisca i vari simboli permetterebbe, inoltre, di implementare una interfaccia di debug efficiente mostrando all'utente l'evoluzione delle tuple all'interno del sistema informativo.
L'analisi semantica sarebbe migliore ed ottimale se si utilizzassero differenti strategie di memorizzazione dei simboli, tuttavia l'implementazione della lista non ordinata permette già una gestione particolareggiata di diverse situazioni di errore e consente, in via embrionale, di:

- controllare le dichiarazioni di variabili e funzioni;
- controllare la correttezza delle inizializzazioni;
- controllare la correttezza degli assegnamenti;
- controllare i tipi di ritorno delle funzioni;
- allocare a runtime le variabili e le funzioni definite dall'utente nel codice;
- deallocare dalla tabella dei simboli i parametri locali delle funzioni.

blocK, di conseguenza, non è in grado ad esempio di valutare la correttezza dei parametri di una chiamata a funzione che potrebbe essere gestita, sempre in maniera non ottimizzata, facendo puntare il record della tabella corrispondente al nome della funzione ad una ulteriore pila contenente gli argomenti.
Immaginando un'ipotetica base di dati associata agli analizzatori, sarebbe sufficiente affiancare alla Symbol Table varie tabelle, tra cui quella delle "Funzioni", per ottenere una mappa chiara ed efficiente dello stato del sistema. Ai fini del controllo e della gestione di eventuali errori basterebbe solamente effettuare delle opportune query contenenti, nel peggiore dei casi, istruzioni di Join.

## *Bibliografia*


- Dispense del corso di Compilatori ed interpreti – Prof. Giacomo Piscitelli
- Dispense del corso di Teoria dei linguaggi – Prof. Giacomo Piscitelli
- http://developer.apple.com/
- http://it.wikipedia.org/wiki/Objective_C
- http://sisinflab.poliba.it/mirizzi/ (per alcuni codici C di esempio)
- Bison 2.4.2, The Yacc-compatible Parser Generator, 24 February 2010, by C. Donnelly and R. Stallman
- Flex manual, by V. Paxson, W. Estes and J. Millaway
- Sviluppare applicazioni con iPhone SDK, di B. Dudney and C. Adamson