Γλώσσες Προγραμματισμού και Μεταγλωττιστές - Εργασία 2

Τσακαλέρης Κωνσταντίνος (dai18071)
dai18071@uom.edu.gr

1, Λεκτικός Αναλυτής

```
%{
#include <stdlib.h>
#include <string.h>
int line = 1;
%}

D [0-9]
LETTER [a-zA-Z]
VARIABLE {LETTER}({LETTER}|{D}|_)*
FLOAT {D}+\.{D}*
newline \n|\x0A|\x0D\x0A
%%
"start" {return T_start;}
"end" {return T_end;}
"print" {return T_print;}
"int" {return T_int;}
"float" {return T_float;}
"abs" {return T_abs;}

"(" {return '(';}
")" {return ')';}
":" {return ':';}
"<" {return '<';}
"=" {return '=';}
">" {return '>';}

("-")?{D}+ {yylval.lexical = strdup(yytext);  return T_num;}
("-")?{FLOAT} {yylval.lexical = strdup(yytext); return
T_real;}
{VARIABLE} {yylval.lexical = strdup(yytext); return T_id;}
```

```
"+" {return '+';}
"-" {return '-';}
"*" {return '*';}
"/" {return '/';}

{newline}  { line++;}
[ \t] { /* nothing */ }
. {
    printf("Lexical Analysis: Unexpected String! :: %s. in
line %d. \n",yytext,yylineno); }
%%
```

Ο λεκτικός αναλυτής είναι ως επί το πλείστον ίδιος με αυτόν που δίνεται με μερικές επιπλέον λεκτικές μονάδες, όπως T_int, T_float, και T_abs, καθώς επίσης και λεκτικές μονάδες που αποτελούνται από ένα μόνο σύμβολο, όπως συγκριτικοί και αριθμητικοί τελεστές.

2. Συντακτικός Αναλυτής

2.1) Tokens και τύποι

```
%token <lexical> T_num
%token <lexical> T_real
%token '('
%token ')'
%token ':'
%token <lexical> T_id
%token <lexical> T_FuncUnary
%token T_start "start"
%token T_end "end"
%token T_print "print"
%token T_int "int"
%token T_float "float"
%token T_abs "abs"

%type<se> expr
%type<condLabels> bool
%type<relopIndex> relop

%left '+' '-' '*' '/'
```

Παραπάνω δηλώνονται οι μονάδες που έρχονται από τον λεκτικό αναλυτή. Επίσης δηλώνονται οι τύποι των σημασιολογικών τιμών των μη-τερματικών συμβόλων expr, bool, relop. Τέλος, οι τελεστές + και * δηλώνονται ως αριστερά προσεταιριστικοί ίδιας προτεραιότητας.

Σημείωση
Δεν είμαι σίγουρος πως έπρεπε να χρησιμοποιήσω το token T_FuncUnary, οπότε δεν το χρησιμοποίησα. Υποθέτω ότι προοριζόταν για χρήση με τα int, float και abs, τα οποία υλοποίησα ως ξεχωριστές μονάδες.
Επίσης δεν είμαι σίγουρος αν έπρεπε να υλοποιήσουμε τις πράξεις της αφαίρεσης και της διαίρεσης, καθώς δεν αναφερόταν κάπου στην εκφώνηση. Συγκεκριμένα, η αφαίρεση προσθέτει ορισμένα ενδεχομένως ανεπιθύμητα features στη γλώσσα τα οποία αναφέρω παρακάτω.

2.2) Κανόνες και σημασιολογικές ρουτίνες:

2.2.1 Ανάθεση τιμής:

```
asmt: T_id expr {
  if (!lookup(symbolTable,$1))
    addvar(&symbolTable,$1,$2.type);
  else
    typeDefinition(lookup_type(symbolTable,$1), $2.type);
  insertSTORE($2.type,lookup_position(symbolTable,$1));
};
```

Μια ανάθεση τιμής αποτελείται από ένα identifier μεταβλητής (T_id) ακολουθούμενο από μία έκφραση. Εάν η μεταβλητή δεν υπάρχει στον πίνακα συμβόλων προστίθεται καινούρια μεταβλητή που έχει τύπο ίδιο με τον τύπο της έκφρασης. Εάν ήδη υπάρχει ελέγχεται ότι ο τύπος της έκφρασης στα δεξιά της μεταβλητής είναι ίδιος με τον ήδη υπάρχοντα τύπο της μεταβλητής. Και στις δύο περιπτώσεις προστίθεται ένα store κατάλληλου τύπου στον ενδιάμεσο κώδικα.

2.2.2 Αριθμητικές εκφράσεις:

```
expr: '(' expr ')' {$$.type = $2.type;}
  | T_num {$$.type = type_integer; pushInteger(atoi($1));}
  | T_real {$$.type = type_real; insertLDC($1);}
```

Αν η έκφραση είναι μία άλλη έκφραση σε παρενθέσεις, τότε ο τύπος της είναι ίδιος με τον τύπο εκείνης της έκφρασης. Σε περίπτωση που η έκφραση είναι μία αριθμητική τιμή (ακέραια ή πραγματική) προστίθεται η αντίστοιχη σταθερά στη στοίβα και η σημασιολογική τιμή της έκφρασης λαμβάνει τον αντίστοιχο τύπο.

```
  | T_id {
      if (!($$.type = lookup_type(symbolTable,$1))) {
        ERR_VAR_MISSING($1,line);
      }
```

```
      insertLOAD($$.type,lookup_position(symbolTable,$1));
    }
```

Αν η έκφραση είναι identifier κάποιας μεταβλητής, γίνεται έλεγχος ότι η μεταβλητή όντως υπάρχει στον πίνακα συμβόλων. Αν υπάρχει, η τιμή της φορτώνεται στη στοίβα με load, αλλιώς παράγεται σφάλμα.

```
  | expr expr '+' {
      $$.type = typeDefinition($1.type, $2.type);
      insertOPERATION($$.type,"add");
    }
  | expr expr '-' {
      $$.type = typeDefinition($1.type, $2.type);
      insertOPERATION($$.type,"sub");
    }
  | expr expr '*' {
      $$.type = typeDefinition($1.type, $2.type);
      insertOPERATION($$.type,"mul");
    }
  | expr expr '/' {
      $$.type = typeDefinition($1.type, $2.type);
      insertOPERATION($$.type,"div");
    }
```

Αν η έκφραση αποτελεί αριθμητική πράξη μεταξύ εκφράσεων, ελέγχεται ότι οι τύποι των εκφράσεων είναι συμβατοί (που εφόσον δεν έχουμε type coercion πρέπει να είναι ίδιοι) και η έκφραση λαμβάνει τον ίδιο τύπο με αυτές. Στη συνέχεια, εισάγεται η αντίστοιχη πράξη μεταξύ των κατάλληλων τύπων στον ενδιάμεσο κώδικα.

```
  | T_id '+' '+' {
      if (!($$.type = lookup_type(symbolTable,$1))) {
        ERR_VAR_MISSING($1,line);
      }
      int position = lookup_position(symbolTable,$1);
      insertLOAD($$.type, position);
      if ($$.type == type_integer)
        insertIINC(position, 1);
      else if ($$.type == type_real)
```

```
        insertFINC(position, 1.0);
    }

  | '+' '+' T_id {
      if (!($$.type = lookup_type(symbolTable,$3))) {
        ERR_VAR_MISSING($3,line);
      }
      int position = lookup_position(symbolTable,$3);
      if ($$.type == type_integer)
        insertIINC(position, 1);
      else if ($$.type == type_real) {
        insertFINC(position, 1.0);
      }
      insertLOAD($$.type, position);
    }
...

%%
void insertFINC(int position, double value) {
  char str_value[MAX_INST_LEN];
  sprintf(str_value, "%f", value);

  insertLOAD(type_real, position);
  insertLDC(str_value);
  insertOPERATION(type_real, "add");
  insertSTORE(type_real,position);
}
```

Αν η έκφραση είναι προσαύξηση κάποιας μεταβλητής αρχικά ελέγχεται αν αυτή υπάρχει στον πίνακα συμβόλων. Ανάλογα με το αν ο τελεστής “++” βρίσκεται πριν ή μετά την μεταβλητή, το inc μπαίνει πριν ή μετά το load της μεταβλητής. Σε περίπτωση που η μεταβλητή είναι τύπου real, επειδή το Java bytecode δεν περιέχει finc instruction, το υλοποίησα με μία ακολουθία από load, ldc, fadd, και store.

Σημείωση
Επέλεξα να χρησιμοποιήσω το MAX_INST_LEN που δηλώνεται στο *jvmLangTypeFunctions.c* για να μετατρέψω τον αριθμό σε αλφαριθμητικό. Για να έχω πρόσβαση στη σταθερά από το bison μετακίνησα το define της στη κεφαλίδα του αρχείου, όποτε χωρίς αυτή την αλλαγή ο κώδικας δεν θα τρέξει.

```
  | "int" expr {
      $$.type = type_integer;
      if ($2.type == type_integer)
        printf("Warning: value is already int, in line
%d.\n",line);
      else
        insertOPERATION($2.type,"2i");
    }
  | "float" expr {
      $$.type = type_real;
      if ($2.type == type_real)
        printf("Warning: value is already real, in line
%d.\n",line);
      else
        insertOPERATION($2.type,"2f");
    }
```

Στις μετατροπές τύπων ελέγχεται ότι ο τύπος της έκφρασης που μετατρέπεται είναι διαφορετικός από τον τύπο στον οποίο μετατρέπεται και εισάγεται η κατάλληλη οδηγία. Αν οι τύποι είναι ίδιοι παράγεται warning.

```
  | expr "abs" {
      $$.type = $1.type;
      insertINVOKESTATIC("java/lang/Math/abs",$1.type,
$1.type);
    }
```

Αν η έκφραση είναι απόλυτη τιμή κάποιας άλλης έκφρασης τότε εισάγεται invoke static με παράμετρο “java/lang/Math/abs” και τους κατάλληλους τύπους.

```
  | bool ':' {
      backpatch($1.trueLbl,currentLabel());
      insertLabel(Label());
    }
    expr ':' {
      backpatch($1.falseLbl,currentLabel());
      $1.nextLbl = makelist(nextInstruction());
```

```
      insertGOTO(UNKNOWN);
      insertLabel(Label());
    }
    expr {
      $$.type = typeDefinition($4.type,$7.type);
      backpatch($1.nextLbl,currentLabel());
      insertLabel(Label());
    };
```

Σε περίπτωση που η έκφραση είναι έκφραση υπό συνθήκη, αρχικά εκτελείται η σημασιολογική ρουτίνα της λογικής έκφρασης bool (που εξηγείται παρακάτω), και έπειτα γίνεται backpatch της trueLbl του συμβόλου bool και εισαγωγή του κατάλληλου label αν η συνθήκη είναι αληθής.
Στη συνέχεια, εκτελείται η σημασιολογική ρουτίνα της πρώτης αριθμητικής έκφρασης, εισάγεται ένα goto προς άγνωστο label το οποίο μπαίνει στο nextLbl του συμβόλου bool (το οποίο πρόσθεσα στον τύπο condLabels), εισάγεται ένα label για τη περίπτωση που η συνθήκη είναι ψευδής και γίνεται backpatch της falseLbl.
Αφού εκτελεστεί και η σημασιολογική ρουτίνα της δεύτερης αριθμητικής έκφρασης, ελέγχεται αν οι τύποι των δύο αριθμητικών εκφράσεων είναι συμβατοί, και γίνεται backpatch της nextLbl με ένα καινούριο label που εισάγεται τώρα.

```
%union{
  ...
  struct {
    NUMBER_LIST_TYPE trueLbl;
    NUMBER_LIST_TYPE falseLbl;
    NUMBER_LIST_TYPE nextLbl;
  } condLabels;
}
```

```
if (. . .) goto Label_True
goto Label_False
Label_True:
. . .
goto Label_Next
Label_False:
. . .
Label_Next:
. . .
```

Οι αλλαγές στον τύπο condLabels και η υλοποίηση του if-else

2.2.3 Λογικές εκφράσεις:

```
bool: expr relop expr {
  typeDefinition($1.type, $3.type);
  if ($1.type == type_integer) {
    $$.trueLbl = makelist(nextInstruction());
    insertICMPOP($2,UNKNOWN);
  }
  if ($1.type == type_real) {
```

```
    insertINSTRUCTION("fcmpl");
    $$.trueLbl = makelist(nextInstruction());
    insertIFOP($2,UNKNOWN);
  }
  $$.falseLbl = makelist(nextInstruction());
  insertGOTO(UNKNOWN);};
```

Αρχικά, γίνεται έλεγχος ότι οι αριθμητικές εκφράσεις που συγκρίνονται είναι του ιδίου τύπου. Στη συνέχεια δημιουργείται η λίστα trueLbl του συμβόλου bool και σε αυτή εισάγεται το κατάλληλο instruction για άλμα υπό συνθήκη ανάλογα με τον τύπο. Τέλος, δημιουργείται η falseLbl και σε αυτή εισάγεται ένα goto προς άγνωστη ετικέτα, το οποίο θα κάνουμε backpatch αργότερα.

2.2.4 Συγκριτικοί Τελεστές:

```
relop: '>'   {$$=OP_GT;}
  | '<' {$$=OP_LT;}
  | '=' {$$=OP_EQ;};
```

3. Παραδείγματα εκτέλεσης

3.1 Simple1:

```
start simple1
(print (3 4 +))
end
```

```
Errors found 0.


 Symbol Table Generated -----


** Compiler Finished!
**********************
** Starting Jasmin Assembler
***************
** Assembler Finished!
**********************
** Executing code in Java
******************
7
```

```
.class public simple1
.super java/lang/Object

.method public static main([Ljava/lang/String;)V
 .limit locals 20
 .limit stack 20

iconst_3
iconst_4
iadd
getstatic java/lang/System/out Ljava/io/PrintStream;
swap
invokevirtual java/io/PrintStream/println(I)V
return
.end method
```

3.2 Simple2:

```
start simple2
(print (3 4 + 7 *))
(print (3.0 4.0 + 7.0 *))
end
```

```
Errors found 0.

 Symbol Table Generated -----

** Compiler Finished!
**********************
** Starting Jasmin Assembler
***************
** Assembler Finished!
**********************
** Executing code in Java
******************
49
49.0
```

```
.class public simple2
.super java/lang/Object

.method public static main([Ljava/lang/String;)V
 .limit locals 20
 .limit stack 20

iconst_3
iconst_4
iadd
bipush 7
imul
getstatic java/lang/System/out Ljava/io/PrintStream;
swap
invokevirtual java/io/PrintStream/println(I)V
ldc 3.0
ldc 4.0
fadd
ldc 7.0
fmul
getstatic java/lang/System/out Ljava/io/PrintStream;
swap
invokevirtual java/io/PrintStream/println(F)V
return
.end method
```

3.3 Simple3:

```
start simple3
(x 1)
(print x)
(print (x++))
(print x)
(y (10))
(print y)
(print (++y))

end
```

```
Errors found 0.

 Symbol Table Generated -----
ST:: Name y of type 1 (i) in Position 2
ST:: Name x of type 1 (i) in Position 1

** Compiler Finished!
**********************
** Starting Jasmin Assembler
***************
** Assembler Finished!
**********************
** Executing code in Java
******************
1
1
2
10
11
```

```
.class public simple3
.super java/lang/Object

.method public static main([Ljava/lang/String;)V
 .limit locals 20
 .limit stack 20

iconst_1
istore 1
iload 1
getstatic java/lang/System/out Ljava/io/PrintStream;
swap
invokevirtual java/io/PrintStream/println(I)V
iload 1
iinc 1 1
getstatic java/lang/System/out Ljava/io/PrintStream;
swap
invokevirtual java/io/PrintStream/println(I)V
iload 1
getstatic java/lang/System/out Ljava/io/PrintStream;
swap
invokevirtual java/io/PrintStream/println(I)V
bipush 10
istore 2
iload 2
getstatic java/lang/System/out Ljava/io/PrintStream;
swap
invokevirtual java/io/PrintStream/println(I)V
iinc 2 1
iload 2
getstatic java/lang/System/out Ljava/io/PrintStream;
swap
invokevirtual java/io/PrintStream/println(I)V
return
.end method
```

3.4 Simple4:

```
start simple4
(y (10))
(x (3 (++y) +))
(print x)
end
```

```
Errors found 0.

 Symbol Table Generated -----
ST:: Name x of type 1 (i) in Position 2
ST:: Name y of type 1 (i) in Position 1

** Compiler Finished!
**********
** Starting Jasmin Assembler
**********
** Assembler Finished!
**********
** Executing code in Java
**********
14
```

```
.class public simple4
.super java/lang/Object

.method public static main([Ljava/lang/String;)V
 .limit locals 20
 .limit stack 20

bipush 10
istore 1
iconst_3
iinc 1 1
iload 1
iadd
istore 2
iload 2
getstatic java/lang/System/out Ljava/io/PrintStream;
swap
invokevirtual java/io/PrintStream/println(I)V
return
.end method
```

3.5 Simple5:

```
start simple5
(x (3 (int 4.0) +))
(print x)
end
```

```
Errors found 0.

 Symbol Table Generated -----
ST:: Name x of type 1 (i) in Position 1

** Compiler Finished!
**********
** Starting Jasmin Assembler
**********
** Assembler Finished!
**********
** Executing code in Java
**********
7
```

```
.class public simple5
.super java/lang/Object

.method public static main([Ljava/lang/String;)V
 .limit locals 20
 .limit stack 20

iconst_3
ldc 4.0
f2i
iadd
istore 1
iload 1
getstatic java/lang/System/out Ljava/io/PrintStream;
swap
invokevirtual java/io/PrintStream/println(I)V
return
.end method
```

3.6 Simple6:

```
start simple6
(x (1))
(x (int (3.0 4.0 +)))
(print x)
end
```

```
Errors found 0.

 Symbol Table Generated -----
ST:: Name x of type 1 (i) in Position 1

** Compiler Finished!
**********
** Starting Jasmin Assembler
**********
** Assembler Finished!
**********
** Executing code in Java
**********
7
```

```
.class public simple6
.super java/lang/Object

.method public static main([Ljava/lang/String;)V
 .limit locals 20
 .limit stack 20

iconst_1
istore 1
ldc 3.0
ldc 4.0
fadd
f2i
istore 1
iload 1
getstatic java/lang/System/out Ljava/io/PrintStream;
swap
invokevirtual java/io/PrintStream/println(I)V
return
.end method
```

3.7 Simple7:

```
start simple7
(x (1.0))
(x (float 3 4 +))
(print x)
(y (1))
(y (int 1 2 + ))
(print y)
end
```

```
Warning: value is already int, in line 6.
Errors found 0.

 Symbol Table Generated -----
ST:: Name y of type 1 (i) in Position 2
ST:: Name x of type 2 (f) in Position 1

** Compiler Finished!
**********************
** Starting Jasmin Assembler
***************
** Assembler Finished!
**********************
** Executing code in Java
******************
7.0
3
```

```
.class public simple7
.super java/lang/Object

.method public static main([Ljava/lang/String;)V
 .limit locals 20
 .limit stack 20

ldc 1.0
fstore 1
iconst_3
iconst_4
iadd
i2f
fstore 1
fload 1
getstatic java/lang/System/out Ljava/io/PrintStream;
swap
invokevirtual java/io/PrintStream/println(F)V
iconst_1
istore 2
iconst_1
iconst_2
iadd
istore 2
iload 2
getstatic java/lang/System/out Ljava/io/PrintStream;
swap
invokevirtual java/io/PrintStream/println(I)V
return
.end method
```

3.8 Simple8:

```
start simple8
(y (-1))
(z (-1.5))
(y (y abs))
(print y)
(z (z abs))
(print z)
end
```

```
Errors found 0.

 Symbol Table Generated -----
ST:: Name z of type 2 (f) in Position 2
ST:: Name y of type 1 (i) in Position 1

** Compiler Finished!
**********************
** Starting Jasmin Assembler
***************
** Assembler Finished!
**********************
** Executing code in Java
******************
1
1.5
```

```
.class public simple8
.super java/lang/Object

.method public static main([Ljava/lang/String;)V
 .limit locals 20
 .limit stack 20

iconst_m1
istore 1
ldc -1.5
fstore 2
iload 1
invokestatic java/lang/Math/abs(I)I
istore 1
iload 1
getstatic java/lang/System/out Ljava/io/PrintStream;
swap
invokevirtual java/io/PrintStream/println(I)V
fload 2
invokestatic java/lang/Math/abs(F)F
fstore 2
fload 2
getstatic java/lang/System/out Ljava/io/PrintStream;
swap
invokevirtual java/io/PrintStream/println(F)V
return
.end method
```

3.9 Simple9:

```
start simple9
(y (1))
(x (0))

(x (y > 0 : 10 : 20))
(print y)
(print x)

(x (y = 0 : 10 : 20))
(print y)
(print x)

end
```

```
backpatch instruction 20 with 1
backpatch instruction 21 with 2
backpatch instruction 24 with 3
backpatch instruction 39 with 4
backpatch instruction 40 with 5
backpatch instruction 43 with 6
Errors found 0.

 Symbol Table Generated -----
ST:: Name x of type 1 (i) in Position 2
ST:: Name y of type 1 (i) in Position 1

** Compiler Finished!
**********
** Starting Jasmin Assembler
**********
** Assembler Finished!
**********
** Executing code in Java
**********
1
10
1
20
```

```
.class public simple9
.super java/lang/Object
.method public static main([Ljava/lang/String;)V
 .limit locals 20
 .limit stack 20
iconst_1
istore 1
iconst_0
istore 2
iload 1
iconst_0
if_icmpgt #_1
goto #_2
#_1:
bipush 10
goto #_3
#_2:
bipush 20
#_3:
istore 2
iload 1
getstatic java/lang/System/out Ljava/io/PrintStream;
swap
invokevirtual java/io/PrintStream/println(I)V
iload 2
getstatic java/lang/System/out Ljava/io/PrintStream;
swap
invokevirtual java/io/PrintStream/println(I)V
iload 1
iconst_0
if_icmpeq #_4
goto #_5
#_4:
bipush 10
goto #_6
#_5:
bipush 20
#_6:
istore 2
iload 1
getstatic java/lang/System/out Ljava/io/PrintStream;
swap
invokevirtual java/io/PrintStream/println(I)V
iload 2
getstatic java/lang/System/out Ljava/io/PrintStream;
swap
invokevirtual java/io/PrintStream/println(I)V
return
.end method
```

3.10 Simple_error:

```
start simple_error
(x 1.0 2)
(y 1)
(y (int 2 3 + ))
(y (z 100 +))
(y (10 20.0 +))
(w 1.0)
(w (float 3.0))
(print x)
(print y)
end
```

```
PARSE ERROR: syntax error, unexpected ')' on line 2.
Warning: value is already int, in line 4.
Variable z NOT declared, n line 5.PARSE ERROR: Variable Declaration fault on line 5.
PARSE ERROR: Type missmatch on line 5.
PARSE ERROR: Type missmatch on line 5.
PARSE ERROR: Type missmatch on line 6.
PARSE ERROR: Type missmatch on line 6.
Warning: value is already real, in line 8.
Variable x NOT declared, n line 9.PARSE ERROR: Variable Declaration fault on line 9.
Errors found 7.
No Code Generated.

 Symbol Table Generated -----
ST:: Name w of type 2 (f) in Position 2
ST:: Name y of type 1 (i) in Position 1
```

3.11 Simple_error2:

```
start simple_error2
(x 1.0)
(x 1)
(print x)
end
```

```
PARSE ERROR: Type missmatch on line 3.
Errors found 1.
No Code Generated.

 Symbol Table Generated -----
ST:: Name x of type 2 (f) in Position 1
```

4. Έξτρα παραδείγματα (δικά μου, λίγο πιο ζόρικα)

4.1 Simple_test:

```
start simple_test
(print 3 4 +)
(print 3.0 4.5 +)
(print 6 4 + 5 *)
(x 15)
(y 2)
(print x y 2 + +)
end
```

```
Errors found 0.

 Symbol Table Generated -----
ST:: Name y of type 1 (i) in Position 2
ST:: Name x of type 1 (i) in Position 1

** Compiler Finished!
**********************
** Starting Jasmin Assembler
***************
** Assembler Finished!
**********************
** Executing code in Java
******************
7
7.5
50
19
```

```
.class public simple_test
.super java/lang/Object

.method public static main([Ljava/lang/String;)V
 .limit locals 20
 .limit stack 20

iconst_3
iconst_4
iadd
getstatic java/lang/System/out Ljava/io/PrintStream;
swap
invokevirtual java/io/PrintStream/println(I)V
ldc 3.0
ldc 4.5
fadd
getstatic java/lang/System/out Ljava/io/PrintStream;
swap
invokevirtual java/io/PrintStream/println(F)V
bipush 6
iconst_4
iadd
iconst_5
imul
getstatic java/lang/System/out Ljava/io/PrintStream;
swap
invokevirtual java/io/PrintStream/println(I)V
bipush 15
istore 1
iconst_2
istore 2
iload 1
iload 2
iconst_2
iadd
iadd
getstatic java/lang/System/out Ljava/io/PrintStream;
swap
invokevirtual java/io/PrintStream/println(I)V
return
.end method
```

4.2 Simple_test2:

```
start simple_test2
(print (3 4 +))
(print (3 4 + 7 *))
(x (22 8 + 2 *) )
(print x)
end
```

```
Errors found 0.

 Symbol Table Generated -----
ST:: Name x of type 1 (i) in Position 1

** Compiler Finished!
**********************
** Starting Jasmin Assembler
***************
** Assembler Finished!
**********************
** Executing code in Java
******************
7
49
60
```

```
.class public simple_test2
.super java/lang/Object

.method public static main([Ljava/lang/String;)V
 .limit locals 20
 .limit stack 20

iconst_3
iconst_4
iadd
getstatic java/lang/System/out Ljava/io/PrintStream;
swap
invokevirtual java/io/PrintStream/println(I)V
iconst_3
iconst_4
iadd
bipush 7
imul
getstatic java/lang/System/out Ljava/io/PrintStream;
swap
invokevirtual java/io/PrintStream/println(I)V
bipush 22
bipush 8
iadd
iconst_2
imul
istore 1
iload 1
getstatic java/lang/System/out Ljava/io/PrintStream;
swap
invokevirtual java/io/PrintStream/println(I)V
return
.end method
```

4.3 Simple_test3 (εμφωλευμένο ternary expression):

```
start simple_test3
(print 1 < 2: 3 > 4: 8 :
16 : 7)
end
```

```
backpatch instruction 16 with 1
backpatch instruction 21 with 2
backpatch instruction 22 with 3
backpatch instruction 25 with 4
backpatch instruction 17 with 5
backpatch instruction 29 with 6
Errors found 0.

 Symbol Table Generated -----

** Compiler Finished!
**********************
** Starting Jasmin Assembler
***************
** Assembler Finished!
**********************
** Executing code in Java
******************
16
```

```
.class public simple_test3
.super java/lang/Object

.method public static main([Ljava/lang/String;)V
 .limit locals 20
 .limit stack 20

iconst_1
iconst_2
if_icmplt #_1
goto #_5
#_1:
iconst_3
iconst_4
if_icmpgt #_2
goto #_3
#_2:
bipush 8
goto #_4
#_3:
bipush 16
#_4:
goto #_6
#_5:
bipush 7
#_6:
getstatic java/lang/System/out Ljava/io/PrintStream;
swap
invokevirtual java/io/PrintStream/println(I)V
return
.end method
```

ισοδύναμο πρόγραμμα:

```
if (1 < 2)
  if (3 > 4)
    return 8;
  else
    return 16;
else
  return 7;
```

4.4 Simple_test4:

```
start simple_test4
(print 8 5 -2 7 - - +)
end
```

```
Errors found 0.

 Symbol Table Generated -----

** Compiler Finished!
**********
** Starting Jasmin Assembler
**********
** Assembler Finished!
**********
** Executing code in Java
**********
22
```

```
.class public simple_test4
.super java/lang/Object

.method public static main([Ljava/lang/String;)V
 .limit locals 20
 .limit stack 20

bipush 8
iconst_5
bipush -2
bipush 7
isub
isub
iadd
getstatic java/lang/System/out Ljava/io/PrintStream;
swap
invokevirtual java/io/PrintStream/println(I)V
return
.end method
```

4.5 Simple_test5:

```
start simple_test5
(print 8 5 -2 7 - / +)
end
```

```
Errors found 0.

 Symbol Table Generated -----

** Compiler Finished!
**********
** Starting Jasmin Assembler
**********
** Assembler Finished!
**********
** Executing code in Java
**********
8
```

```
.class public simple_test5
.super java/lang/Object

.method public static main([Ljava/lang/String;)V
 .limit locals 20
 .limit stack 20

bipush 8
iconst_5
bipush -2
bipush 7
isub
idiv
iadd
getstatic java/lang/System/out Ljava/io/PrintStream;
swap
invokevirtual java/io/PrintStream/println(I)V
return
.end method
```

4.6 Simple_test6:
(Η σωστή απάντηση είναι -3.8, αλλά περιορίζεται από την ακρίβεια των floats)

```
start simple_test6
(print 28.0 5.0 / (float
-2) 7.4 - +)
end
```

```
Errors found 0.

 Symbol Table Generated -----

** Compiler Finished!
**********************
** Starting Jasmin Assembler
***************
** Assembler Finished!
**********************
** Executing code in Java
******************
-3.7999997
```

```
.class public simple_test6
.super java/lang/Object

.method public static main([Ljava/lang/String;)V
 .limit locals 20
 .limit stack 20

ldc 28.0
ldc 5.0
fdiv
bipush -2
i2f
ldc 7.4
fsub
fadd
getstatic java/lang/System/out Ljava/io/PrintStream;
swap
invokevirtual java/io/PrintStream/println(F)V
return
.end method
```

4.7 Simple_test7:
Για να εκτελεστεί η αφαίρεση σωστά θα πρέπει να υπάρχει κενό ανάμεσα στο πλην και στο δύο, ώστε να γίνει διαφοροποίηση από το -2. Ευτυχώς, αυτό δεν επηρεάζει την γλώσσα, καθώς αν θέλαμε να χρησιμοποιήσουμε το -2, θα χρειαζόμασταν έναν επιπλέον τελεστή.

```
start simple_test7
(print 8 5 - 2 7 - +)
end
```

```
Errors found 0.

 Symbol Table Generated -----

** Compiler Finished!
**********************
** Starting Jasmin Assembler
***************
** Assembler Finished!
**********************
** Executing code in Java
******************
-2
```

```
.class public simple_test7
.super java/lang/Object

.method public static main([Ljava/lang/String;)V
 .limit locals 20
 .limit stack 20

bipush 8
iconst_5
isub
iconst_2
bipush 7
isub
iadd
getstatic java/lang/System/out Ljava/io/PrintStream;
swap
invokevirtual java/io/PrintStream/println(I)V
return
.end method
```