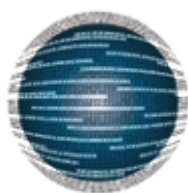


**Εργαστηριακή Άσκηση
Αρχές Γλωσσών Προγραμματισμού
και Μεταφραστών**

Project Flex-Bison

Ακαδημαϊκό Έτος 2023-24



Ον/μο: Κουτρομπέλας Βασίλειος

ΑΜ: 1093397

Έτος: 3^ο

E-mail: up1093397@ac.upatras.gr

Ον/μο: Μινώπετρος Φίλιππος

ΑΜ: 1093431

Έτος: 3^ο

E-mail: up1093431@ac.upatras.gr

Εισαγωγή.....	3
Ερώτημα 1ο.....	4
Περιγραφή της γραμματικής της γλώσσας σε BNF.....	4
Σχόλια στον κώδικα Flex/Bison.....	13
Flex.....	13
Bison.....	16
Παραδείγματα Εκτέλεσης.....	17
Ερώτημα 2ο.....	19
Περιγραφή της γραμματικής της γλώσσας σε BNF.....	19
Σχόλια στον κώδικα Flex/Bison.....	20
Flex.....	20
Bison.....	20
Παραδείγματα Εκτέλεσης.....	21
Ερώτημα 3ο.....	22
Περιγραφή της γραμματικής της γλώσσας σε BNF.....	22
Σχόλια στον κώδικα Flex/Bison.....	25
Flex.....	25
Bison.....	25
Παραδείγματα Εκτέλεσης.....	29
Ερώτημα 4ο.....	32
Περιγραφή της γραμματικής της γλώσσας σε BNF.....	32
Σχόλια στον κώδικα Flex/Bison.....	32
Flex.....	32
Bison.....	32
Παραδείγματα Εκτέλεσης.....	33
Παράρτημα.....	34
Παραδοχές.....	34
Κώδικας.....	34
Ερώτημα 1ο.....	34
BNF.....	34
Flex.....	37
Bison.....	39
Ερώτημα 2ο.....	46
BNF.....	46
Flex.....	50
Bison.....	52
Ερώτημα 3ο.....	59
BNF.....	59
Flex.....	63
Bison.....	65
Ερώτημα 4ο.....	79
BNF.....	79
Flex.....	83
Bison.....	85

Εισαγωγή

Το παρόν έγγραφο αποτελεί την αφορά για την εργαστηριακή άσκηση του μαθήματος Αρχές Γλωσσών Προγραμματισμού και Μεταφραστών. Σχολιάζεται, η περιγραφή μιας φανταστικής αντικειμενοστραφούς γλώσσας προγραμματισμού σε BNF, η υλοποίηση ενός λεκτικού καθώς και ενός συντακτικού αναλυτή, χρησιμοποιώντας τα εργαλεία Flex και Bison. Ως ομάδα εργαστήκαμε παράλληλα και ασύγχρονα, δια ζώσης και εξ αποστάσεως, με τα κύρια εργαλεία που αξιοποιήσαμε να είναι το [GitHub](#) και το [Live Share](#) extension του Visual Studio Code που μας επέτρεπε να επεξεργαζόμαστε σε ζωντανό χρόνο τα ίδια αρχεία.

Ερώτημα 1^ο

Περιγραφή της γραμματικής της γλώσσας σε BNF

```
<CHAR> ::= '\\' <letter> '\\'
<STRING> ::= '"' ( <letter> | <digit> | [ -~ ] )* '"'
<INT> ::= <digit>+
<DOUBLE> ::= <digit>+ "." <digit>+
<CLASS_IDENTIFIER> ::= [A-Z] ( <letter> | <digit> | "_" )*
<IDENTIFIER> ::= <letter> ( <letter> | <digit> | "_" )*

<digit> ::= "0" | ... | "9"
<letter> ::= "a" | ... | "z" | "A" | ... | "Z"
```

Για αρχή στην γραμματική ορίζουμε σε κανόνες τους αριθμούς και γράμματα ώστε να τα χρησιμοποιήσουμε σε κανόνες των primitive και non-primitive τύπων δεδομένων που θα μας χρειαστούν.

```
<program> ::= <class_declarations>
```

Το πρόγραμμα του συντακτικού αναλυτή ξεκινάει με τον κανόνα program που αποτελείται μόνο από τον κανόνα δήλωσης κλάσεων (class_declarations), καθώς όπως ορίζουν και οι προδιαγραφές τα προγράμματα της γλώσσας που υλοποιούμε οργανώνονται σε μία ή περισσότερες κλάσεις.

```

<class_declarations> ::= <class_declaration>
                        | <class_declarations> <class_declaration>

<class_declaration> ::= <modifier> "class" <CLASS_IDENTIFIER> "{" <class_body> "}"

<modifier> ::= "public"
              | "private"

```

Ο κανόνας `class_declarations` που αποτελεί και το κορμό του προγράμματος μπορεί να περιέχει μία δήλωση κλάσης (`class_declaration`) ή πολλές δηλώσεις αναδρομικά.

Η κλάση δηλώνεται ακολουθώντας την παραπάνω δομή(`class_declaration`), με το `modifier` να μπορεί να είναι είτε `public` είτε `private`, τη λέξη `class`, το όνομα της κλάσης και το σώμα της που εμπεριέχεται σε αγκύλες. Επίσης, πιο πριν έχει οριστεί ο κανόνας `CLASS_IDENTIFIER` ώστε το πρώτο γράμμα του ονόματος της κλάσης να είναι κεφαλαίο και να μπορούν να χρησιμοποιηθεί η κάτω παύλα.

Παράδειγμα:

```
public class Car {...}
```

```

<class_body> ::= ε
              | <class_body_elements>

<class_body_elements> ::= <class_body_element>
                        | <class_body_elements> <class_body_element>

<class_body_element> ::= <declaration>
                        | <method_declaration>
                        | <class_declaration>
                        | <assignment>

```

Κάθε κλάση μπορεί να περιέχει ένα ή περισσότερα στοιχεία εκ των οποίων αυτά μπορούν να είναι προαιρετικά, δηλώσεις μεταβλητών και ανάθεση σε αυτές, δηλώσεις μεθόδων και δημιουργία εμφωλευμένων κλάσεων.

```

<declaration> ::= <data_type> <IDENTIFIER> ";"
                | <modifier> <data_type> <IDENTIFIER> ";"

<data_type> ::= "int"
                | "char"
                | "double"
                | "boolean"
                | "void"
                | "String"
                | <CLASS_IDENTIFIER>

```

Οι μεταβλητές, οι μέθοδοι και τα αντικείμενα μπορούν να δηλωθούν βάση του κανόνα declaration, με προαιρετικό modifier, τον τύπο τους, το όνομά τους και ερωτηματικό.

Σύμφωνα με τον κανόνα data_types μπορούν να είναι ακέραιοι, χαρακτήρες, δεκαδικοί, Boolean ή συμβολοσειρές, ενώ τα void και CLASS_IDENTIFIER έχουν προστεθεί αποκλειστικά για τη δημιουργία μεθόδων και αντικειμένων αντίστοιχα.

Παράδειγμα:

```

int my_int;
public String my_string="Hello World!";

```

```

<method_declaration> ::= <modifier> <data_type> <IDENTIFIER> "(" <parameter_list> ")" "{" <block> <return_stmt> "}"

<parameter_list> ::= ε
                  | <parameters>

<parameters> ::= <parameter>
                | <parameters> "," <parameter>

<parameter> ::= <data_type> <IDENTIFIER>

```

Μια μέθοδος ορίζεται με modifier, τον τύπο που επιστρέφει, το όνομα της και την λίστα παραμέτρων στην οποία προαιρετικά δέχεται ένα, κανένα ή πολλά ονόματα μεταβλητών χωρισμένα με κόμμα. Κάθε στοιχείο της λίστας αποτελείται από τον τύπο δεδομένων και το όνομα της μεταβλητής. Έπειτα εντός των αγκυλών τοποθετείται το επιθυμητό μπλοκ κώδικα συνοδευόμενο από το return statement τα οποία περιγράφονται παρακάτω.

```
<method_call> ::= <IDENTIFIER> "(" <identifier_list> ")" ";"  
  
<identifier_list> ::= ε  
                    | <identifiers>  
  
<identifiers> ::= <IDENTIFIER>  
                | <identifiers> "," <IDENTIFIER>  
                | <member_access>
```

Η κλήση μεθόδου γίνεται γράφοντας το όνομα της μεθόδου και σε παρένθεση την λίστα ορισμάτων την οποία μπορεί να δέχεται. Η λίστα ορισμάτων λειτουργεί με τρόπο όπως αυτή των παραμέτρων αλλά χωρίς την δήλωση τύπου δεδομένων.

```
<member_access> ::= <IDENTIFIER> "." <IDENTIFIER>  
                  | <IDENTIFIER> "." <method_call>
```

Για την πρόσβαση στα μέλη των κλάσεων και των μεθόδων χρησιμοποιείται ο κανόνας `member_access`, με το όνομα του αντικειμένου, τελεία και το όνομα το μέλους ή την κλήση της μεθόδου.

```

<block> ::= <statement>
        | <block> <statement>

<statement> ::= <declaration>
               | <method_call>
               | <assignment>
               | <dowhile>
               | <for>
               | <if>
               | <switch>
               | "break" ";"
               | <print> ";"

```

Τα block αποτελούνται από ένα ή περισσότερα statement αναδρομικά. Τα statement μπορούν να είναι: δήλωση, κλήση μεθόδου, ανάθεση, do-while loop, for loop, έλεγχος, switch, εκτύπωση και break με ερωτηματικό.

```

<assignment> ::= <IDENTIFIER> "=" <assigned_value> ";"
               | <IDENTIFIER> "=" "new" <CLASS_IDENTIFIER> "(" <identifier_list> ")" ";"
               | <member_access> "=" <assigned_value> ";"
               | <IDENTIFIER> "=" <method_call>

<assigned_value> ::= <expression>
                  | <CHAR>
                  | <STRING>
                  | "true"
                  | "false"

```

Σε μία ανάθεση (assignment) μπορεί μία μεταβλητή ή ένα μέλος να λάβει απευθείας τιμή ή με την κλήση μεθόδου. Στα αντικείμενα η ανάθεση μπορεί επίσης να είναι η δημιουργία νέου στιγμιότυπου μίας κλάσης με τη λέξη new, το όνομα της κλάσης, και τα ορίσματα σε παρένθεση.

Οι τιμές που μπορούν να ανατεθούν αναγράφονται στον κανόνα assigned_value και μπορούν να είναι ολόκληρες εκφράσεις, χαρακτήρας, συμβολοσειρά, true και false.


```
<return_stmt> ::= "return" <assigned_value> ";"  
                | "return" ";"
```

Το return τοποθετείται συνήθως στο τέλος μιας μεθόδου δηλαδή μετά από ένα block κώδικα και επιστρέφει είτε μια τιμή που μπορεί να ανατεθεί σε μεταβλητή, είτε αν ο τύπος επιστροφής της μεθόδου είναι void, δείχνει απλα το τέλος της μεθόδου.

```
<expression> ::= <term>  
                | <expression> "-" <term>  
                | <expression> "+" <term>  
  
<term> ::= <factor>  
            | <term> "*" <factor>  
            | <term> "/" <factor>  
  
<factor> ::= <IDENTIFIER>  
            | <INT>  
            | "-" <INT>  
            | <DOUBLE>  
            | "-" <DOUBLE>  
            | "(" <expression> ")"  
            | <member_access>
```

Η παραπάνω δομή ακολουθεί την προτεραιότητα των πράξεων. Ξεκινώντας την ανάλυσή της, μία έκφραση μπορεί να είναι είτε κάποιος όρος, είτε έκφραση μείον ή συν κάποιον όρο.

Ένας όρος μπορεί να είναι είτε ένας παράγοντας, είτε όρος επί ή διά έναν παράγοντα.

Οι όροι μπορούν να είναι: μεταβλητή ή αντικείμενο, θετικός ή αρνητικός ακέραιος, θετικός ή αρνητικός δεκαδικός, έκφραση σε παρένθεση ή κάποιο μέλος.

```

<condition> ::= <assigned_value>
              | <assigned_value> <logic_operator> <assigned_value>

<logic_operator> ::= "=="
                    | "!="
                    | ">"
                    | "<"
                    | "&&"
                    | "||"

```

Μία προϋπόθεση (condition) μπορεί να είναι κάποια τιμή(assigned_value), όχι απαραίτητα αριθμός, ή μία τιμή σε σύγκριση(logic_operator) με μία άλλη.

Οι συγκρίσεις με τη σειρά που αναγράφονται στον κανόνα logic_operator είναι: ίσο, διάφορο, μεγαλύτερο, μικρότερο, λογικό 'και', λογικό 'ή'.

```

<dowhile> ::= "do" "{" <block> "}" "while" "(" <condition> ")" ";"

```

Η δομή ενός do-while loop είναι: η λέξη do, το σώμα της επανάληψης μέσα σε αγκύλες, η λέξη while, η προϋπόθεση μέσα σε παρενθέσεις και ερωτηματικό.

Παράδειγμα:

```
do{
```

```
...
```

```
}while(kilometers<5);
```

```

<for> ::= "for" "(" <assignment> <condition> ";" <assignment> ")" "{" <block> "}"
        | "for" "(" <assignment> <condition> ";" <assignment> ")" <statement>

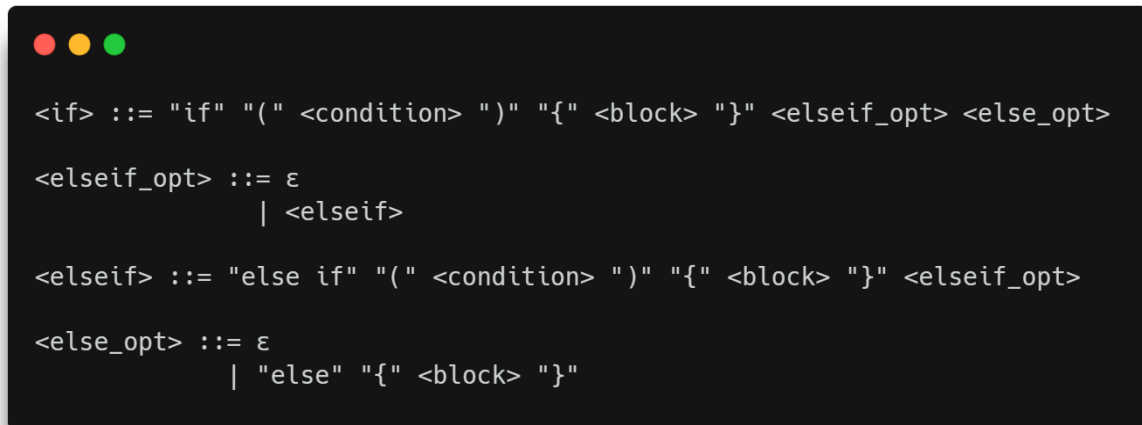
```

Ένα for loop αποτελείται από τη λέξη for, μία ανάθεση(assignment) μαζί με μία προϋπόθεση(condition) και άλλη μία ανάθεση μέσα σε παρένθεση. Στη συνέχεια μπορεί να έχει είτε ένα statement είτε ένα block μέσα σε αγκύλες.

Παράδειγμα:

```
for(int i=0; i<12; i=i+1;) method1();
```

```
for(int i=0; i<12; i=i+1;){  
    ...  
}
```



```
<if> ::= "if" "(" <condition> ")" "{" <block> "}" <elseif_opt> <else_opt>  
<elseif_opt> ::= ε  
                | <elseif>  
<elseif> ::= "else if" "(" <condition> ")" "{" <block> "}" <elseif_opt>  
<else_opt> ::= ε  
                | "else" "{" <block> "}"
```

Ένας έλεγχος if δομείται με: τη λέξη if, μία προϋπόθεση(condition) μέσα σε παρενθέσεις, block μέσα σε αγκύλες, προαιρετικό else-if (elseif_opt) και προαιρετικό else (else_opt).

Για να είναι ο κανόνας elseif_opt προαιρετικός, μπορεί να είναι είτε κενός , είτε ένα elseif.

Ο κανόνας elseif: ξεκινάει με τις λέξεις else if, μία προϋπόθεση σε παρενθέσεις, ένα block μέσα σε αγκύλες και ένα προαιρετικό else-if, ώστε να μπορούν να υπάρχουν πάνω από ένα.

Τέλος ο κανόνας else_opt ακολουθεί τη λογική του elseif_opt, έτσι μπορεί να είναι είτε κενός, είτε η λέξη else και ένα block μέσα σε αγκύλες.

Παράδειγμα:

```
if(kilometers==10){  
    ...  
}else if(kilometers>10{  
    ...  
}|else{  
    ...  
}
```

```

<switch> ::= "switch" "(" <expression> ")" "{" <case_blocks> <default_block_opt> "}"

<case_blocks> ::= <case_block>
                | <case_blocks> <case_block>

<case_block> ::= "case" <expression> ":" "{" <block> "}"

<default_block_opt> ::= ε
                    | "default" ":" "{" <block> "}"

```

Ο κανόνας switch ακολουθεί τη δομή: η λέξη switch, μία έκφραση μέσα σε παρενθέσεις, case_blocks και ένα προαιρετικό default block (default_block_opt) μέσα σε αγκύλες.

Στη συνέχεια, τα case_blocks είναι είτε ένα case_block είτε περισσότερα αναδρομικά.

Το case_block αποτελείται από τη λέξη case, μία έκφραση, άνω και κάτω τελεία και ένα block μέσα σε αγκύλες.

Το default_block_opt ακολουθεί τη δομή των προηγούμενων προαιρετικών κανόνων, έτσι μπορεί να είναι είτε κενό είτε η λέξη default, άνω και κάτω τελεία και ένα block μέσα σε αγκύλες.

Παράδειγμα:

```
switch(selection)
```

```
{
    case '1':
        {
            ...
        }
    case '2':
        {
            ...
        }
    default:
        {
            ...
        }
}
```

```
<print> ::= "out.print" "(" <STRING> ")"
          | "out.print" "(" <STRING> "," <identifier_list> ")"
```

Το print ορίζεται ως: οι λέξεις out.print και είτε μία συμβολοσειρά (STRING) μέσα σε παρενθέσεις, είτε μία συμβολοσειρά και ένα identifier_list μέσα σε παρενθέσεις χωρισμένα με κόμμα.

Σχόλια στον κώδικα Flex/Bison

Flex

Στο κομμάτι του Flex υλοποιήσαμε την αναγνώριση ονόματος κλάσης, μεταβλητών αλλά και όλων των τύπων δεδομένων που χρησιμοποιούνται στην γλώσσα και την επιστροφή αντίστοιχων token.

```
digit [0-9]
letter [a-zA-Z]
...
[A-Z]({letter}|{digit}|_)* { yylval.str = strdup(yytext); return
CLASS_IDENTIFIER; }
{letter}({letter}|{digit}|_)* { yylval.str = strdup(yytext);
return IDENTIFIER; }

{digit}+ { yylval.intval = atoi(yytext); return INT; }
{digit}+"."{digit}+ { yylval.dblval = atof(yytext); return
DOUBLE; }

\'.\' {yylval.charval = yytext[0]; return CHAR;}
\"({letter}|{digit}|[ -~])*\" { yylval.str = strdup(yytext);
return STRING; }
```

Επίσης, επιστρέφει token δεσμευμένων λέξεων που δεν μπορούν να χρησιμοποιηθούν ως κοινά αναγνωριστικά.

```
"int" { return INT_DATA_TYPE; }
"char" { return CHAR_DATA_TYPE; }
"double" { return DOUBLE_DATA_TYPE; }
"boolean" { return BOOLEAN_DATA_TYPE; }
"void" { return VOID_DATA_TYPE; }
"String" { return STRING_DATA_TYPE; }

"true" { return BOOLEAN_TRUE; }
"false" { return BOOLEAN_FALSE; }

"public" { return PUBLIC; }
"private" { return PRIVATE; }

"new" { return NEW; }
"class" { return CLASS; }
"return" { return RETURN; }
"break" { return BREAK; }
"do" { return DO; }
"while" { return WHILE; }
"for" { return FOR; }
"if" { return IF; }
"else" { return ELSE; }
"else if" { return ELSEIF; }
"switch" { return SWITCH; }
"case" { return CASE; }
"default" { return DEFAULT; }
"out.print" { return PRINT; }
```

Επιστρέφει ειδικούς χαρακτήρες που χρησιμοποιούνται από την γραμματική (Bison) για την ροή του προγράμματος ή ως τελεστές λογικών πράξεων.

```

"==" { return EQ_OP; }
"!=" { return NEQ_OP; }
">" { return GT_OP; }
"<" { return LT_OP; }
"&&" { return AND_OP; }
"||" { return OR_OP; }

";" { return SEMICOLON; }
"," { return COMMA; }
"(" { return LPAREN; }
")" { return RPAREN; }
"{" { return LBRACE; }
"}" { return RBRACE; }
"." { return DOT; }
"=" { return ASSIGN; }
"+" { return PLUS; }
"-" { return MINUS; }
"*" { return MULT; }
"/" { return DIV; }
":" { return COLON; }

```

Τέλος, για την αναγνώριση σχολίων μόλις αναγνωστούν “//” αγνοείται η υπόλοιπη γραμμή μιας και ο λεκτικός αναλυτής δεν επιστρέφει κανένα token μέχρι να διαβάσει \newline, ενώ μόλις αναγνωστούν οι χαρακτήρες “/*” αγνοείται ότι ακολουθεί (συμπεριλαμβανομένου και των \newline) μέχρι να αναγνωστούν οι χαρακτήρες “*/”

```

"//".* { /* ignore comments */ }
"/*"(.| [ -~^* / ] | ( \n ))*"*/" { /* ignore comments */ }

```

Bison

Στον κώδικα του Bison λαμβάνονται τα token από το Flex και σε όσα χρειάζεται ορίζεται ο τύπος δεδομένων τους χρησιμοποιώντας το union. Στη main γίνεται το άνοιγμα του δοκιμαστικού αρχείου με τον κώδικα της γλώσσας, η ζητούμενη εκτύπωση στο terminal του συνόλου του κώδικα, η κλήση του yyparse, καθώς και η εμφάνιση της σωστής χρήσης του parser από τον χρήστη σε περίπτωση που δώσει λάθος argument. Όσον αφορά συναρτήσεις έχει οριστεί μόνο η yyerror με τρόπο τέτοιο ώστε να τυπώνει μήνυμα λάθους στην οθόνη καθώς και τη γραμμή στην οποία βρέθηκε. Στην υπόλοιπη έκταση του κώδικα γίνεται εφαρμογή της γραμματικής BNF όπως περιγράφηκε στο αντίστοιχο κεφάλαιο με τις κατάλληλες αλλαγές για να λειτουργεί σε γλώσσα Bison.

```
int main(int argc, char** argv) {
    if (argc == 2) {
        yyin = fopen(argv[1], "r");
        if (!yyin) {
            perror(argv[1]);
            return 1;
        }

        //PRINT SOURCE CODE=====
        FILE* file_copy = fopen(argv[1], "r");
        char c = fgetc(file_copy);
        while (c != EOF)
        {
            printf ("%c", c);
            c = fgetc(file_copy);
        }
        fclose(file_copy);
        //=====

        yyparse();
    }
    else
        printf("Usage: %s <filename>\n", argv[0]);
    return 0;
}
```

```
%union {
    int intval;
    double dblval;
    char *str;
    char charval;
}

%token <intval> INT
%token <dblval> DOUBLE
%token <charval> CHAR
%token <str> IDENTIFIER CLASS_IDENTIFIER
STRING
...
```

```
void yyerror(const char *s) {
    fprintf(stderr, "Error: %s in line %d\n", s, yylineno);
}
```


Παραδείγματα Εκτέλεσης

```
./a.out test.txt
public class A
{
    int integer;
    //NESTED CLASS
    private class B
    {
        int data;
    }
    char character;
    String string;
    boolean bool;
    public void method1(Object object)
    {
        data = -9.3;
        string="thats a string";
        bool = true;
        character='c';
        //OBJECT
        Object1 object1;
        object1 = new Object(ident1, ident2, ident3);
        //MEMBER ACCESS
        a.integer=b.whatever+4;
        //ASSIGN METHOD CALL
        data = method1(object);
        method1(object);

        do{
            int a;
            a=5+5;
            //SUB EXAMPLE
            a= 4-4;
        }while(a>0);
    }
}
```

```
//FOR LOOP SINGLE LINE
for(i=0; i<10; i=i+1;) a=8;
//FOR LOOP MULTIPLE LINES
for(i=0; i<10; i=i+1;){
    //IF MULTIPLE LINES ELSE IF ELSE
    if(i==5)
    {
        a=6;
    }
    else if(i==6)
    {
        a=6;
    }
    else if(i==7)
    {
        a=7 ;
    }else
    {
        a=0;
        break;
    }
}

num=1;
//SWITCH CASE
switch(num){
    case 1: {counter=100; break;}
    case 2: {counter=101; break;}
    default: {counter=102; break;}
}

//SWITCH CASE NO DEFAULT
switch(num)
{
    case 1: {counter=100; break;}
}

//PRINT
out.print("counter now: %d", counter.attack);

return;
}
}
-- Successful Parse :D --
```

Π1] Επιτυχής εκτέλεση κώδικα που αναδεικνύει τη γραμματική που αναγνωρίζεται στο ερώτημα 1. Στο πρώτο βήμα βλέπουμε την κλήση του προγράμματος για να ελέγξει το αρχείο test.txt, στη συνέχεια βλέπουμε τον κώδικα που διαβάστηκε και απο τη στιγμή που δεν υπήρξε κάποιο λάθος στο τέλος υπάρχει το μήνυμα επιβεβαίωσης.

```

./a.out test_error.txt
public class A
{
    char String character;
    String string;
    boolean bool;
    public void method1(Object object)
    {
        data = -9.3;
        string="thats a string";
        bool = true;
        character='c';
        //OBJECT
        Object1 object1;
        object1 = new Object(ident1, ident2, ident3);
        //MEMBER ACCESS
        a.integer=b.whatever+4;
        //ASSIGN METHOD CALL
        data = method1(object);
        method1(object);
        return;
    }
}
Error: syntax error, unexpected STRING_DATA_TYPE,
expecting IDENTIFIER in line 3

```

Π2]

Στο συγκεκριμένο παράδειγμα υπάρχει λάθος δήλωση μεταβλητής, αρχικά τυπώνεται ο κώδικας και στη συνέχεια εμφανίζεται το κατάλληλο μήνυμα για το λάθος καθώς και η γραμμή στην οποία βρίσκεται. Επίσης η εκτέλεση σταματά στο πρώτο error.

```

> ./a.out
Usage: ./a.out <filename>

```

Π3]

Αν δεν περάσουμε τα κατάλληλα ορίσματα κατά την εκτέλεση του προγράμματος, όπως το αρχείο με τον κώδικα προς ανάλυση, εμφανίζεται στην οθόνη οδηγία για την ορθή χρήση.

Ερώτημα 2^ο

Στο συγκεκριμένο κεφάλαιο σχολιάζονται μόνο τα επιμέρους κομμάτια του κώδικα που άλλαξαν ή προστέθηκαν σε εκείνον του αντίστοιχου κεφαλαίου του ερωτήματος 1.

Περιγραφή της γραμματικής της γλώσσας σε BNF

```
<declaration> ::= <data_type> <IDENTIFIER> ";"  
                | <modifier> <data_type> <IDENTIFIER> ";"  
                | <data_type> <IDENTIFIER> "=" <assigned_value> ";" //[1]  
                | <modifier> <data_type> <IDENTIFIER> "=" <assigned_value> ";" //[2]  
                | <data_type> <identifier_list> ";" //[3]  
                | <data_type> <assignment_list> ";" //[4]  
  
<assignment_list> ::= <IDENTIFIER> "=" <assigned_value>  
                    | <assignment_list> "," <IDENTIFIER> "=" <assigned_value>
```

Με πράσινο σημειώνονται οι προσθήκες. Όσον αφορά τον κανόνα declaration τα [1] και [2] είναι για τη δήλωση μεταβλητής και ανάθεση τιμής σε αυτή σε μία ενιαία εντολή, με προαιρετικό modifier. Τέλος, τα [3] και [4] είναι για πολλαπλές δηλώσεις και πολλαπλές δηλώσεις με αναθέσεις ίδιου τύπου δεδομένων, αντίστοιχα.

Ο κανόνας assignment_list είναι συμπληρωματικός του παραπάνω υποστηρίζοντας μία ή πολλαπλές αναθέσεις χωρισμένες με κόμμα.

Παράδειγμα:

[1] int data=3;

[2] public int data=3;

[3] int a, b, c, d, e;

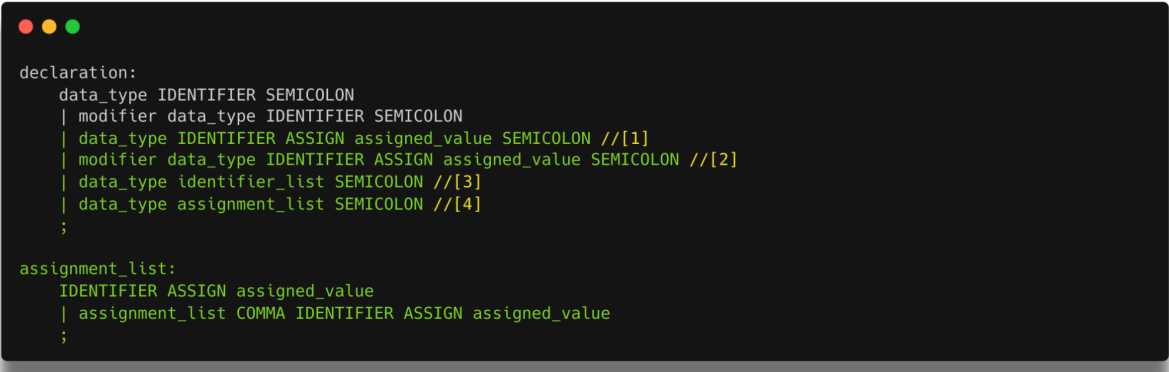
[4] int a=1, b=2, c=3, d=4, e=5;

Σχόλια στον κώδικα Flex/Bison

Flex

Δεν χρειάστηκαν προσαρμογές στον κώδικα του Flex.

Bison



```
declaration:
  data_type IDENTIFIER SEMICOLON
  | modifier data_type IDENTIFIER SEMICOLON
  | data_type IDENTIFIER ASSIGN assigned_value SEMICOLON //[1]
  | modifier data_type IDENTIFIER ASSIGN assigned_value SEMICOLON //[2]
  | data_type identifier_list SEMICOLON //[3]
  | data_type assignment_list SEMICOLON //[4]
  ;

assignment_list:
  IDENTIFIER ASSIGN assigned_value
  | assignment_list COMMA IDENTIFIER ASSIGN assigned_value
  ;
```

Με πράσινο σημειώνονται οι προσθήκες και είναι απλή εφαρμογή της γραμματικής BNF σε κώδικα Bison.

Παραδείγματα Εκτέλεσης

```
./a.out test.txt
public class A
{
    int integer;
    integer=10;
    //NESTED CLASS
    private class B
    {
        int data;
    }
    char character;
    String string;
    boolean bool;
    public void method1(Object object)
    {
        data = -9.3;
        string="thats a string";
        bool = true;
        character='c';
        //OBJECT
        Object1 object1;
        object1 = new Object(ident1, ident2, ident3);
        //MEMBER ACCESS
        a.integer=b.whatever+4;
        //ASSIGN METHOD CALL
        data = method1(object);
        method1(object);

        do{
            int a;
            a=5+5;
            //SUB EXAMPLE
            a= 4-4;
        }while(a>0);

        //FOR LOOP SINGLE LINE
        for(i=0; i<10; i=i+1;) a=8;
        //FOR LOOP MULTIPLE LINES
        for(i=0; i<10; i=i+1;){
            //IF MULTIPLE LINES ELSE IF ELSE
            if(i==5)
            {
                a=6;
            }
            else if(i==6)
            {
                a=6;
            }
        }
    }
}
```

```
        else if(i==7)
        {
            a=7 ;
        }else
        {
            a=0;
            break;
        }
    }

    num=1;
    //SWITCH CASE
    switch(num){
        case 1: {counter=100; break;}
        case 2: {counter=101; break;}
        default: {counter=102; break;}
    }

    //SWITCH CASE NO DEFAULT
    switch(num)
    {
        case 1: {counter=100; break;}
    }

    //PRINT
    out.print("counter now: %d", counter.attack);

    //DECLARATION WITH ASSIGNMENT
    public String my_str="Test";
    String str="Test";

    //MULTIPLE DECLARATIONS AND ASSIGNMENTS
    int x = 15, y = 16, z = 17;
    double x, y, z;

    return;
}
-- Successful Parse :D --
```

Παράδειγμα μιας επιτυχούς εκτέλεσης. Επισημαίνονται με μπλε οι αλλαγές, δηλαδή ότι πλέον επιτρέπονται οι πολλαπλές δηλώσεις και οι πολλαπλές δηλώσεις με αναθέσεις.

Ερώτημα 3^ο

Στο συγκεκριμένο κεφάλαιο σχολιάζονται μόνο τα επιμέρους κομμάτια του κώδικα που άλλαξαν ή προστέθηκαν σε εκείνον του αντίστοιχου κεφαλαίου του ερωτήματος 2.

Περιγραφή της γραμματικής της γλώσσας σε BNF

```
<assignment> ::= <IDENTIFIER> "=" <assigned_value> ";"
                | <IDENTIFIER> "=" <expression> ";"
                | <IDENTIFIER> "=" "new" <CLASS_IDENTIFIER> "(" <identifier_list> ")" ";"
                | <member_access> "=" <assigned_value> ";"
                | <member_access> "=" <method_call> ";"
                | <IDENTIFIER> "=" <method_call>

<assigned_value> ::= <CHAR>
                  | <STRING>
                  | "true"
                  | "false"
```

Αρχικά, το expression διαχωρίστηκε από το assigned_value και μετατράπηκε σε ξεχωριστό κανόνα ανάθεσης στο assignment, ώστε να μπορούν να εκτελούνται σωστά οι αριθμητικές πράξεις. αλλιώς δεν θα μπορούσε να γίνει σωστή διαχείριση της διαδικασίας. Ακολούθως οι αλλαγές στην υπόλοιπη γραμματική οι οποίες είναι απαραίτητες μετά την παραπάνω αλλαγή.

```
<assignment_list> ::= <IDENTIFIER> "=" <assigned_value>
                    | <assignment_list> "," <IDENTIFIER> "=" <assigned_value>
                    | <IDENTIFIER> "=" <expression>
                    | <assignment_list> "," <IDENTIFIER> "=" <expression>
```

Στον κανόνα assignment_list έγιναν οι απαραίτητες αλλαγές ώστε να δέχεται και την ανάθεση εκφράσεων σε πολλαπλές αναθέσεις.

Παράδειγμα:

```
int a=1, b=3+1;
```

```

<declaration> ::= <data_type> <IDENTIFIER> ";"
                | <modifier> <data_type> <IDENTIFIER> ";"
                | <data_type> <IDENTIFIER> "=" <assigned_value> ";"
                | <modifier> <data_type> <IDENTIFIER> "=" <assigned_value> ";"
                | <data_type> <IDENTIFIER> "=" <expression> ";"
                | <modifier> <data_type> <IDENTIFIER> "=" <expression> ";"
                | <data_type> <identifier_list> ";"
                | <data_type> <assignment_list> ";"

```

Στον κανόνα declaration προστέθηκαν δύο επιπλέον σεντ από token που αφορούν την ταυτόχρονη δήλωση μεταβλητής και ανάθεση τιμής σε αυτήν. Οι προσθήκες αυτές είναι απαραίτητες λόγω του διαχωρισμού του expression από τον κανόνα assigned_value.

```

<condition> ::= <assigned_value>
                | <condition> <logic_operator> <assigned_value>
                | <expression>
                | <condition> <logic_operator> <expression>

```

Στις συνθήκες προστέθηκε η δυνατότητα χρήσης έκφρασης ή σύγκρισης μίας οποιασδήποτε συνθήκης με κάποια έκφραση. Επίσης, μία λεπτομέρεια είναι ότι χρησιμοποιήθηκε αναδρομή στο τμήμα <condition> που σε προηγούμενη εκδοχή ήταν <assigned_value>.

Παράδειγμα:

if(a<1) ...

if(a<(b+2)) ...

```
<method_call_list> ::= "(" ">
                        | <method_identifiers> ")"

<method_identifiers> ::= "(" <IDENTIFIER>
                        | "(" <member_access>
                        | <method_identifiers> "," <IDENTIFIER>
                        | <method_identifiers> "," <member_access>
```

Επιπλέον, έγινε προσθήκη του κανόνα `method_identifiers` με σκοπό οι μέθοδοι να μπορούν να δέχονται ως ορίσματα πέρα από μεταβλητές, και μέλη. Έτσι, έγινε και η κατάλληλη προσαρμογή του `method_call_list` ώστε να μπορεί να είναι είτε κενό είτε να δέχεται ένα ή περισσότερα ορίσματα.

Παράδειγμα:

```
method1();
method2(my_var);
method3(my_var, my_object.member, my_int);
```


Σχόλια στον κώδικα Flex/Bison

Flex

Δεν υπήρξαν αλλαγές.

Bison

Ο κανόνες σε γλώσσα Bison αποτελούν απλή εφαρμογή της γραμματικής BNF που σχολιάστηκε προηγουμένως. Συνεπώς, στο συγκεκριμένο τμήμα γίνεται λόγος κυρίως για το μέρος σε γλώσσα C που συνδυάστηκε με τη γραμματική για την υλοποίηση των ερωτημάτων.

```
typedef struct identifier
{
    char* name;
    int block_level;
    double value;
} Identifier;

typedef struct node
{
    Identifier *data;
    struct node *next;
} Node

Node *method_head;
Node *identifier_head;
```

Για την υλοποίηση των λειτουργιών: εκτέλεση πράξεων, αποθήκευση τιμής μεταβλητής και τον έλεγχο του scope των identifier υλοποιήθηκαν 2 linked list, ένα για τις μεθόδους και ένα για τα υπόλοιπα identifiers. Τα linked list ακολουθούν τη δομή των struct που παρουσιάζονται στο απόσπασμα.

Πιο συγκεκριμένα στο Node, το next pointer δείχνει στο επόμενο Node και το data pointer χρησιμοποιείται για την αποθήκευση δεδομένων.

Στο struct Identifier, αποθηκεύεται το όνομα της μεταβλητής, ή αντικειμένου, το βάθος του block στο οποίο δηλώθηκε, καθώς και την τιμή αν έχει γίνει ανάθεση. Τέλος, η τιμή, χάριν απλότητας είναι τύπου double.

Για διευκόλυνση ανάγνωσης, στις συναρτήσεις C που ακολουθούν δίνεται μόνο η δήλωσή τους, ενώ το σώμα τους υπάρχει στο κεφάλαιο **Παράρτημα** μαζί με τον υπόλοιπο κώδικα του ερωτήματος.

```

// Function to create a new node
Node* createNode(char* name, int block_level);

// Function to insert a node at the end of the list
void insertNode(Node **head, char* name, int block_level);

// Function to print the linked list
void printList(Node *head);

// Function to free memory allocated for the linked list
void freeList(Node *head);

```

Ξεκινώντας την ανάλυση των συναρτήσεων, υλοποιήθηκαν κάποιες βοηθητικές συναρτήσεις για την διαχείριση των λιστών, η `insertNode`, που χρησιμοποιείται για να εισάγει ένα `Node` στο τέλος της λίστας που παίρνει ως όρισμα και παράλληλα καλεί την `createNode` που εκτελεί τις κατάλληλες ενέργειες για την δέσμευση μνήμης για το καινούριο `Node`. Στη συνέχεια η συνάρτηση `freeList` αξιοποιείται μετά την εκτέλεση του parsing για ελευθέρωση της μνήμης που δεσμεύτηκαν από τα list (`identifier_head`, `method_head`). Η `printList` υπάρχει για λόγους debugging να εμφανίζονται όλα τα περιεχόμενα μίας λίστας.

```

// Function that searches for an identifier and checks if it has been declared and is
// in the right scope
void searchErrors(Node *head, char* name);

// Function that searches and returns the value of an identifier
double searchIdentifier(Node *head, char* name);

// Function that performs an operation via the pop_operand function and gets the
// result to the identifier struct
double getDataToIdentifier(Node *head, char* name);

// Function that deletes from the identifier list all identifiers that are out of
// scope
void scope_collapse(Node** head, int current_block);

```

Για την υλοποίηση του Variable Scope χρησιμοποιείται η global int μεταβλητή `current_block` η οποία ορίζει το βάθος ενός block και αυξάνεται με την ύπαρξη αριστερής αγκύλης “{” και μειώνεται με την ύπαρξη δεξιάς “}”. Με κάθε δήλωση μεταβλητής, κλάσης, αντικειμένου ή μεθόδου, αποθηκεύεται η τιμή του τρέχον block και όταν ένα block κλείνει χρησιμοποιείται η συνάρτηση `scope_collapse`, η οποία διατρέχει μία λίστα και διαγράφει με ασφάλεια τα

Node που το block depth τους είναι ίσο με το current_block ώστε να αποφευχθεί να μπορεί να ζητηθεί μία μεταβλητή out of scope. Η συνάρτηση αυτή τρέχει για μία από τις δύο λίστες ή και για τις δύο ανάλογα τον τύπο του block. Η searchErrors είναι εκείνη που αναλαμβάνει τον έλεγχο και την εμφάνιση των κατάλληλων μηνυμάτων μη-δήλωσης ή δήλωσης out of scope. Η searchIdentifier χρησιμοποιείται για την αναζήτηση μίας μεταβλητής σε λίστα και την επιστροφή της τιμής της αν αυτή υπάρχει, ενώ η getDataToIdentifier αναζητά μια μεταβλητή και εκτελεί αριθμητική πράξη.

```
void searchErrors(Node *head, char* name) {
    Node *current = head;

    while (current != NULL) {
        if (strcmp(current->data->name, name) == 0) {
            return;
        }
        current = current->next;
    }
    printf("Error: identifier \"%s\" in line %d not declared in this scope.\n", name,
        yylineno);
    exit(EXIT_FAILURE);
}
```

Η searchErrors πέρα των λειτουργιών που αναφέρθηκαν προηγουμένως, όταν εντοπιστεί error κάνει έξοδο από το πρόγραμμα (exit(EXIT_FAILURE), όπου EXIT_FAILURE=1).

```
double operand_stack[MAX_STACK_SIZE];
char operator_stack[MAX_STACK_SIZE];
int operand_top = -1;
int operator_top = -1;

void push_operand(double value);


void push_operator(char op);

double pop_operand();

char pop_operator();

double perform_operation(char op, double val1, double val2);
```

Για τις αριθμητικές πράξεις αξιοποιήσαμε τον [Dijkstra's Two-Stack Algorithm](#) που χρησιμοποιεί ένα stack για τους τελεστές(operator_stack) και ένα για τους τελεστέους(operand_stack). Η συνάρτηση push_operator ευθύνεται για την αποθήκευση των συμβόλων των πράξεων στο αντίστοιχο stack, ενώ η push_operand για την αποθήκευση των τιμών. Η push_operator καλεί την pop_operator, 2 φορές την pop_operand για στην συνέχεια την push_operand με όρισμα την perform_operation η οποία εκτελεί την πράξη και αποθηκεύεται το αποτέλεσμα στο stack. Κατάλληλα μηνύματα εμφανίζονται σε περίπτωση Overflow και Underflow των Stacks.



```
void printSource(const char* file_name){
```

Η συνάρτηση printSource αξιοποιείται για την εμφάνιση του πηγαίου κώδικα εισόδου. Δέχεται ως όρισμα το argv[1] μέσα στη main, λαμβάνει ένα προς ένα τους χαρακτήρες από το αρχείο και όσο δεν εντοπίζει το EOF τους εμφανίζει στην οθόνη και προχωράει στον επόμενο χαρακτήρα.

Παραδείγματα Εκτέλεσης

Π1]

Παράδειγμα με ανάθεση τιμής σε μεταβλητή που δεν έχει δηλωθεί.

```
./a.out test_a1.txt
___Start of Source Code___
public class A
{
    public void method1()
    {
        my_var=4;
        return;
    }
}
___End of Source Code___
Error: identifier "my_var" in line 5 not declared in this scope.
```

Π2]

Κλήση μη-ορισμένης μεθόδου.

```
./a.out test_a2.txt
___Start of Source Code___
public class A
{
    public void method1()
    {
        method2();
        return;
    }
}
___End of Source Code___
Error: identifier "method2" in line 5 not declared in this scope.
```

Π3]

Ανάθεση τιμής σε μεταβλητή που έχει δηλωθεί σε διαφορετικό scope.

```
./a.out test_b1.txt
___Start of Source Code___
public class A
{
    public void method1()
    {
        int my_var;
        my_var=3;
        return;
    }
    public void method2()
    {
        my_var=4;
        return;
    }
}
___End of Source Code___
my_var=3.00
Error: identifier "my_var" in line 11 not declared in this scope.
```

```

./a.out test_b2.txt
___Start of Source Code___
public class A
{
    private class B{
        private void b_method(){
            int a=4;
            return;
        }
    }

    private void a_method(){
        b_method();
    }
}
___End of Source Code___
a=4.00
Error: identifier "b_method" in line 11 not declared in this scope.

```

Π4]

Κλήση μεθόδου που έχει οριστεί σε διαφορετική κλάση.

```

./a.out test_c.txt
___Start of Source Code___
public class A
{
    public void method1()
    {
        int a=4;
        int b=2;
        int c=6;
        int d=9;
        double result1 = a+b*c/d; // result must be 5.33
        int result2 = ((a+b)*c)/d; // result must be 4
        return;
    }
}
___End of Source Code___
a=4.00
b=2.00
c=6.00
d=9.00
result1=5.33
result2=4.00
-- Successful Parse :D --

```

Π5]

Αριθμητικές εκφράσεις με επίδειξη προτεραιότητας πράξεων και παρενθέσεων.

```
./a.out test_c.txt
___Start of Source Code___
public class A
{
    public void method1()
    {
        int a=4;
        int b=2;
        int c=6;
        int d=9;

        double result1 = a+b*c/d; // result must be 5.33
        int result2 = ((a+b)*c)/d; // result must be 4
        double result3=a+e;
        return;
    }
}
___End of Source Code___
a=4.00
b=2.00
c=6.00
d=9.00
result1=5.33
result2=4.00
Error: identifier "e" in line 12 not declared in this scope.
```

Π6]

Αριθμητική έκφραση σε περίπτωση που δεν έχει οριστεί μία από τις μεταβλητές που συμμετέχουν.

Ερώτημα 4^ο

Περιγραφή της γραμματικής της γλώσσας σε BNF

Δεν υπήρξαν αλλαγές στη γραμματική σε σχέση με το ερώτημα 3.

Σχόλια στον κώδικα Flex/Bison

Flex

Δεν υπήρξαν αλλαγές.

Bison

```
int searchErrors(Node *head, char* name) {
    Node *current = head;

    while (current != NULL) {
        if (strcmp(current->data->name, name) == 0) {
            return 0;
        }
        current = current->next;
    }
    printf("Error: identifier \"%s\" in line %d not declared in this scope.\n", name,
yylineno);
    return 1;
}
```

Το κομμάτι που χρειάστηκε αλλαγή σε σχέση με το ερώτημα 3 ήταν η αλλαγή τύπου επιστροφής της συνάρτησης `searchErrors` από `void` σε `int` και η αντικατάσταση του `exit` με εντολές `return`, με την κατάλληλη τιμή 0 (μη-εντοπισμός λάθους) ή 1 (εντοπισμός λάθους), ώστε να γίνεται εμφάνιση όλων των λαθών του προγράμματος εισόδου και να μην γίνεται έξοδος στο πρώτο λάθος. Επίσης, όταν ζητείται η τιμή μίας μεταβλητής, πρώτα καλείται η `searchErrors` και μετά η συνάρτηση επιστροφής τιμής της μεταβλητής, μόνο αν δεν βρεθεί λάθος.

Απόσπασμα από τον κώδικα Bison:

```
IDENTIFIER ASSIGN expression SEMICOLON {  
    if(!searchErrors(identifier_head, $1, current_block)){  
        double result=getDataToIdentifier(identifier_head, $1);  
        printf("%s=%.2lf\n", $1, result);  
    }  
}
```

Παραδείγματα Εκτέλεσης

```
//parser version 4  
  
./a.out test_final.txt  
___Start of Source Code___  
public class A{  
    int a=-4;  
    int b=8;  
  
    //variable c is not declared  
    int result1= a+b+c;  
  
    int result2= a+b;  
  
    private void method1(int myvar, char mychar){  
        double mydouble;  
        return;  
    }  
    //variable out of scope  
    mydouble=4.1;  
    double snik_the_hustla=21.9;  
}  
___End of Source Code___  
a=-4.00  
b=8.00  
Error: identifier "c" in line 6 not declared in this scope.  
result1=4.00  
result2=4.00  
Error: identifier "mydouble" in line 15 not declared in this scope.  
snik_the_hustla=21.90  
-- Successful Parse :D --
```

```
//parser version 3  
  
a.out test_final.txt  
___Start of Source Code___  
public class A{  
    int a=-4;  
    int b=8;  
  
    //variable c is not declared  
    int result1= a+b+c;  
  
    int result2= a+b;  
  
    private void method1(int myvar, char mychar){  
        double mydouble;  
        return;  
    }  
    //variable out of scope  
    mydouble=4.1;  
    double snik_the_hustla=21.9;  
}  
___End of Source Code___  
a=-4.00  
b=8.00  
Error: identifier "c" in line 6 not declared in this scope.
```

Σύγκριση μεταξύ των parser των ερωτημάτων 3 και 4. Εκείνος του ερωτήματος 3 σταματάει την εκτέλεση στο πρώτο λάθος, ενώ του ερωτήματος 4 συνεχίζει και εμφανίζει όλα τα λάθη στον κώδικα.

Παράρτημα

Παραδοχές

1] Για τα ερωτήματα 3 & 4: Τυπώνονται στην οθόνη όλες οι αναθέσεις που αφορούν expression ενώ θα έπρεπε να εμφανίζονται μόνο τα αποτελέσματα από αριθμητικές πράξεις. Ωστόσο, δεν έγινε η συγκεκριμένη τροποποίηση λόγω πολυπλοκότητας της γραμματικής.

2] Δεν υλοποιήθηκε η αποθήκευση τιμών σε μέλη κατά την προσβασιμότητα (πχ. myobject.myvar=1).

3] Για όλα τα αριθμητικά χρησιμοποιείται double και αυτό έγινε για διατήρηση της απλότητας, αλλά είναι δυνατή η προσαρμογή του κώδικα ώστε να ανταποκρίνεται σε int κλπ.

Κώδικας

Ερώτημα 1^ο

BNF

<CHAR> ::= "\" <letter> "\"

<STRING> ::= "\"" (<letter> | <digit> | [~\]) * "\""

<INT> ::= <digit> +

<DOUBLE> ::= <digit> + "." <digit> +

<CLASS_IDENTIFIER> ::= [A-Z] (<letter> | <digit> | "_") *

<IDENTIFIER> ::= <letter> (<letter> | <digit> | "_") *

<digit> ::= "0" | ... | "9"

<letter> ::= "a" | ... | "z" | "A" | ... | "Z"

<program> ::= <class_declarations>

<class_declarations> ::= <class_declaration>

| <class_declarations> <class_declaration>

<class_declaration> ::= <modifier> "class" <CLASS_IDENTIFIER> "{" <class_body> "}"

<modifier> ::= "public"

| "private"

<class_body> ::= ε

| <class_body_elements>

<class_body_elements> ::= <class_body_element>
| <class_body_elements> <class_body_element>

<class_body_element> ::= <declaration>
| <method_declaration>
| <class_declaration>
| <assignment>

<declaration> ::= <data_type> <IDENTIFIER> ";"
| <modifier> <data_type> <IDENTIFIER> "·;"

<data_type> ::= "int"
| "char"
| "double"
| "boolean"
| "void"
| "String"
| <CLASS_IDENTIFIER>

<method_declaration> ::= <modifier> <data_type> <IDENTIFIER> "(" <parameter_list> ")"
"{" <block> <return_stmt> "}"

<parameter_list> ::= ε
| <parameters>

<parameters> ::= <parameter>
| <parameters> "," <parameter>

<parameter> ::= <data_type> <IDENTIFIER>

<method_call> ::= <IDENTIFIER> "(" <identifier_list> ")" ";"

<identifier_list> ::= ε
| <identifiers>

<identifiers> ::= <IDENTIFIER>
| <identifiers> "," <IDENTIFIER>
| <member_access>

<member_access> ::= <IDENTIFIER> "." <IDENTIFIER>
| <IDENTIFIER> "." <method_call>

<block> ::= <statement>
| <block> <statement>

<statement> ::= <declaration>

- | <method_call>
- | <assignment>
- | <dowhile>
- | <for>
- | <if>
- | <switch>
- | "break" ";"
- | <print> ";"

<assignment> ::= <IDENTIFIER> "=" <assigned_value> ";"

- | <IDENTIFIER> "=" "new" <CLASS_IDENTIFIER> "(" <identifier_list> ")" ";"
- | <member_access> "=" <assigned_value> ";"
- | <member_access> "=" <method_call> ";"
- | <IDENTIFIER> "=" <method_call>

<assigned_value> ::= <expression>

- | <CHAR>
- | <STRING>
- | "true"
- | "false"

<expression> ::= <term>

- | <expression> "-" <term>
- | <expression> "+" <term>

<term> ::= <factor>

- | <term> "*" <factor>
- | <term> "/" <factor>

<factor> ::= <IDENTIFIER>

- | <INT>
- | "-" <INT>
- | <DOUBLE>
- | "-" <DOUBLE>
- | "(" <expression> ")"
- | <member_access>

<condition> ::= <assigned_value>

- | <assigned_value> <logic_operator> <assigned_value>

<logic_operator> ::= "=="

- | "!="

```

| ">"
| "<"
| "&&"
| "||"

```

`<dowhile> ::= "do" "{" <block> "}" "while" "(" <condition> ")" ";"`

`<for> ::= "for" "(" <assignment> <condition> ";" <assignment> ")" "{" <block> "}"
| "for" "(" <assignment> <condition> ";" <assignment> ")" <statement>`

`<if> ::= "if" "(" <condition> ")" "{" <block> "}" <elseif_opt> <else_opt>`

`<elseif_opt> ::= ε
| <elseif>`

`<elseif> ::= "else if" "(" <condition> ")" "{" <block> "}" <elseif_opt>`

`<else_opt> ::= ε
| "else" "{" <block> "}"`

`<switch> ::= "switch" "(" <expression> ")" "{" <case_blocks> <default_block_opt> "}"`

`<case_blocks> ::= <case_block>
| <case_blocks> <case_block>`

`<case_block> ::= "case" <expression> ":" "{" <block> "}"`

`<default_block_opt> ::= ε
| "default" ":" "{" <block> "}"`

`<print> ::= "out.print" "(" <STRING> ")"
| "out.print" "(" <STRING> "," <identifier_list> ")"`

`<return_stmt> ::= "return" <assigned_value> ";"
| "return" ";"`

Flex

%{

```

#include <stdio.h>
#include <stdlib.h>
#include "parser.tab.h"
%}
%option yylineno

digit [0-9]
letter [a-zA-Z]

%%

"int" { return INT_DATA_TYPE; }
"char" { return CHAR_DATA_TYPE; }
"double" { return DOUBLE_DATA_TYPE; }
"boolean" { return BOOLEAN_DATA_TYPE; }
"void" { return VOID_DATA_TYPE; }
"String" { return STRING_DATA_TYPE; }

"true" { return BOOLEAN_TRUE; }
"false" { return BOOLEAN_FALSE; }

"public" { return PUBLIC; }
"private" { return PRIVATE; }

"new" { return NEW; }
"class" { return CLASS; }
"return" { return RETURN; }
"break" { return BREAK; }
"do" { return DO; }
"while" { return WHILE; }
"for" { return FOR; }
"if" { return IF; }
"else" { return ELSE; }
"else if" { return ELSEIF; }
"switch" { return SWITCH; }
"case" { return CASE; }
"default" { return DEFAULT; }
"out.print" { return PRINT; }

"==" { return EQ_OP; }
"!=" { return NEQ_OP; }
">" { return GT_OP; }
"<" { return LT_OP; }
"&&" { return AND_OP; }
"||" { return OR_OP; }

```

```

";" { return SEMICOLON; }
"," { return COMMA; }
"(" { return LPAREN; }
")" { return RPAREN; }
"{" { return LBRACE; }
"}" { return RBRACE; }
"." { return DOT; }
"=" { return ASSIGN; }
"+" { return PLUS; }
"-" { return MINUS; }
"*" { return MULT; }
"/" { return DIV; }
":" { return COLON; }

[A-Z]({letter}|{digit}|_)* { yylval.str = strdup(yytext); return CLASS_IDENTIFIER; }
{letter}({letter}|{digit}|_)* { yylval.str = strdup(yytext); return IDENTIFIER; }

{digit}+ { yylval.intval = atoi(yytext); return INT; }
{digit}+"."{digit}+ { yylval.dblval = atof(yytext); return DOUBLE; }

\' \' { yylval.charval = yytext[0]; return CHAR; }
\"({letter}|{digit}|[ ~])*\" { yylval.str = strdup(yytext); return STRING; }

\"/\".* { /* ignore comments */ }
\"/*\"([ ~^*\/]|(\n))*\"/*\" { /* ignore comments */ }

[ \t\n]+ { /* ignore whitespace */ }

%%

int yywrap() {
    return 1;
}

```

Bison

```

%{
#include <stdio.h>
#include <stdlib.h>

```

```

#include <string.h>
extern FILE *yyin;

extern int yylineno;
void yyerror(const char *s);
int yylex(void);

%}

%define parse.error verbose

%union {
    int intval;
    double dblval;
    char *str;
    char charval;
}

%token <intval> INT
%token <dblval> DOUBLE
%token <charval> CHAR
%token <str> IDENTIFIER CLASS_IDENTIFIER STRING

%token INT_DATA_TYPE CHAR_DATA_TYPE DOUBLE_DATA_TYPE
BOOLEAN_DATA_TYPE VOID_DATA_TYPE STRING_DATA_TYPE
%token BOOLEAN_TRUE BOOLEAN_FALSE
%token PUBLIC PRIVATE
%token NEW CLASS RETURN BREAK DO WHILE FOR IF ELSE ELSEIF SWITCH
CASE DEFAULT PRINT
%token SEMICOLON COMMA LPAREN RPAREN LBRACE RBRACE DOT ASSIGN
PLUS MINUS MULT DIV COLON
%token EQ_OP NEQ_OP GT_OP LT_OP AND_OP OR_OP

%%

program:
    class_declarations { printf("-- Successful Parse :D --\n"); }
    ;

class_declarations:
    class_declaration
    | class_declarations class_declaration
    ;

```



```
class_declaration:
    modifier CLASS CLASS_IDENTIFIER LBRACE class_body RBRACE
    ;
```

```
modifier:
    PUBLIC
    | PRIVATE
    ;
```

```
class_body:
    /* empty */
    | class_body_elements
    ;
```

```
class_body_elements:
    class_body_element
    | class_body_elements class_body_element
    ;
```

```
class_body_element:
    declaration
    | method_declaration
    | class_declaration
    | assignment
    ;
```

```
declaration:
    data_type IDENTIFIER SEMICOLON
    | modifier data_type IDENTIFIER SEMICOLON
    ;
```

```
data_type:
    INT_DATA_TYPE
    | CHAR_DATA_TYPE
    | DOUBLE_DATA_TYPE
    | BOOLEAN_DATA_TYPE
    | VOID_DATA_TYPE
    | STRING_DATA_TYPE
    | CLASS_IDENTIFIER
    ;
```

```
method_declaration:
```

```
    modifier data_type IDENTIFIER LPAREN parameter_list RPAREN LBRACE block
return_stmt RBRACE
;
```

```
method_call:
    IDENTIFIER LPAREN identifier_list RPAREN SEMICOLON
```

```
parameter_list:
    /* empty */
    | parameters
    ;
```

```
parameters:
    parameter
    | parameters COMMA parameter
    ;
```

```
parameter:
    data_type IDENTIFIER
    ;
```

```
identifier_list:
    /* empty */
    | identifiers
    ;
```

```
identifiers:
    IDENTIFIER
    | identifiers COMMA IDENTIFIER
    | member_access
    ;
```

```
member_access:
    IDENTIFIER DOT IDENTIFIER;
    | IDENTIFIER DOT method_call
    ;
```

```
block:
    statement
    | block statement
    ;
```

```
statement:
    declaration
```

- | method_call
- | assignment
- | dowhile
- | for
- | if
- | switch
- | BREAK SEMICOLON
- | print SEMICOLON

;

assignment:

- IDENTIFIER ASSIGN assigned_value SEMICOLON
- | IDENTIFIER ASSIGN NEW CLASS_IDENTIFIER LPAREN identifier_list RPAREN SEMICOLON
- | member_access ASSIGN assigned_value SEMICOLON
- | member_access ASSIGN method_call SEMICOLON
- | IDENTIFIER ASSIGN method_call

;

assigned_value:

- expression
- | CHAR
- | STRING
- | BOOLEAN_TRUE
- | BOOLEAN_FALSE

;

expression:

- term
- | expression MINUS term
- | expression PLUS term

;

term:

- factor
- | term MULT factor
- | term DIV factor

;

factor:

- IDENTIFIER
- | INT
- | MINUS INT
- | DOUBLE

```
| MINUS DOUBLE
| LPAREN expression RPAREN
| member_access
;
```

```
condition:
    assigned_value
    | assigned_value logic_operator assigned_value
;
```

```
logic_operator:
    EQ_OP
    | NEQ_OP
    | GT_OP
    | LT_OP
    | AND_OP
    | OR_OP
;
```

```
dowhile:
    DO LBRACE block RBRACE WHILE LPAREN condition RPAREN SEMICOLON
;
```

```
for:
    FOR LPAREN assignment condition SEMICOLON assignment RPAREN LBRACE block
    RBRACE
    | FOR LPAREN assignment condition SEMICOLON assignment RPAREN statement
;
```

```
if:
    IF LPAREN condition RPAREN LBRACE block RBRACE elseif_opt else_opt
;
```

```
elseif_opt:
    /* empty */
    | elseif
;
```

```
elseif:
    ELSEIF LPAREN condition RPAREN LBRACE block RBRACE elseif_opt
;
```

```
else_opt:
    /* empty */
```

```

    | ELSE LBRACE block RBACE
;

switch:
    SWITCH LPAREN expression RPAREN LBRACE case_blocks default_block_opt
    RBACE
;

case_blocks:
    case_block
    | case_blocks case_block
;

case_block:
    CASE expression COLON LBRACE block RBACE
;

default_block_opt:
    /* empty */
    | DEFAULT COLON LBRACE block RBACE
;

print:
    PRINT LPAREN STRING RPAREN
    | PRINT LPAREN STRING COMMA identifier_list RPAREN
;

return_stmt:
    RETURN assigned_value SEMICOLON
    | RETURN SEMICOLON
;

%%

void yyerror(const char *s) {
    fprintf(stderr, "Error: %s in line %d\n", s, yylineno);
}

int main(int argc, char** argv) {
    if (argc == 2) {
        yyin = fopen(argv[1], "r");
        if (!yyin) {
            perror(argv[1]);
            return 1;
        }
    }
}

```

```

    }

    //PRINT SOURCE CODE
    FILE* file_copy = fopen(argv[1], "r");
    char c = fgetc(file_copy);
    while (c != EOF)
    {
        printf("%c", c);
        c = fgetc(file_copy);
    }
    fclose(file_copy);

    yyparse();
}
else
    printf("Usage: %s <filename>\n", argv[0]);
return 0;
}

```

Ερώτημα 2°

BNF

```

<CHAR> ::= "\" <letter> "\"
<STRING> ::= "'" ( <letter> | <digit> | [ ~\ ] ) * "'"
<INT> ::= <digit> +
<DOUBLE> ::= <digit> + "." <digit> +
<CLASS_IDENTIFIER> ::= [A-Z] ( <letter> | <digit> | "_" ) *

```

```

<IDENTIFIER> ::= <letter> ( <letter> | <digit> | "_" ) *

<digit> ::= "0" | ... | "9"
<letter> ::= "a" | ... | "z" | "A" | ... | "Z"

<program> ::= <class_declarations>

<class_declarations> ::= <class_declaration>
                        | <class_declarations> <class_declaration>

<class_declaration> ::= <modifier> "class" <CLASS_IDENTIFIER> "{" <class_body> "}"

<modifier> ::= "public"
              | "private"

<class_body> ::= ε
              | <class_body_elements>

<class_body_elements> ::= <class_body_element>
                        | <class_body_elements> <class_body_element>

<class_body_element> ::= <declaration>
                        | <method_declaration>
                        | <class_declaration>
                        | <assignment>

<declaration> ::= <data_type> <IDENTIFIER> ";"
                | <modifier> <data_type> <IDENTIFIER> ";"
                | <data_type> <IDENTIFIER> "=" <assigned_value> ";"
                | <modifier> <data_type> <IDENTIFIER> "=" <assigned_value> ";"
                | <data_type> <identifier_list> ";"
                | <data_type> <assignment_list> ";"

<assignment_list> ::= <IDENTIFIER> "=" <assigned_value>
                    | <assignment_list> "," <IDENTIFIER> "=" <assigned_value>

<data_type> ::= "int"
              | "char"
              | "double"
              | "boolean"
              | "void"
              | "String"
              | <CLASS_IDENTIFIER>

```

<method_declaration> ::= <modifier> <data_type> <IDENTIFIER> "(" <parameter_list> ")"
"{" <block> <return_stmt> "}"

<parameter_list> ::= ϵ
| <parameters>

<parameters> ::= <parameter>
| <parameters> "," <parameter>

<parameter> ::= <data_type> <IDENTIFIER>

<method_call> ::= <IDENTIFIER> "(" <identifier_list> ")" ";"

<identifier_list> ::= ϵ
| <identifiers>

<identifiers> ::= <IDENTIFIER>
| <identifiers> "," <IDENTIFIER>
| <member_access>

<member_access> ::= <IDENTIFIER> "." <IDENTIFIER>
| <IDENTIFIER> "." <method_call>

<block> ::= <statement>
| <block> <statement>

<statement> ::= <declaration>
| <method_call>
| <assignment>
| <dowhile>
| <for>
| <if>
| <switch>
| "break" ";"
| <print> ";"

<assignment> ::= <IDENTIFIER> "=" <assigned_value> ";"
| <IDENTIFIER> "=" "new" <CLASS_IDENTIFIER> "(" <identifier_list> ")" ";"
| <member_access> "=" <assigned_value> ";"
| <member_access> "=" <method_call> ";"
| <IDENTIFIER> "=" <method_call>

<assigned_value> ::= <expression>
| <CHAR>

- | <STRING>
- | "true"
- | "false"

<expression> ::= <term>
 | <expression> "-" <term>
 | <expression> "+" <term>

<term> ::= <factor>
 | <term> "*" <factor>
 | <term> "/" <factor>

<factor> ::= <IDENTIFIER>
 | <INT>
 | "-" <INT>
 | <DOUBLE>
 | "-" <DOUBLE>
 | "(" <expression> ")"
 | <member_access>

<condition> ::= <assigned_value>
 | <assigned_value> <logic_operator> <assigned_value>

<logic_operator> ::= "=="
 | "!="
 | ">"
 | "<"
 | "&&"
 | "||"

<dowhile> ::= "do" "{" <block> "}" "while" "(" <condition> ")" ";"

<for> ::= "for" "(" <assignment> <condition> ";" <assignment> ")" "{" <block> "}"
 | "for" "(" <assignment> <condition> ";" <assignment> ")" <statement>

<if> ::= "if" "(" <condition> ")" "{" <block> "}" <elseif_opt> <else_opt>

<elseif_opt> ::= ϵ
 | <elseif>

<elseif> ::= "else if" "(" <condition> ")" "{" <block> "}" <elseif_opt>

<else_opt> ::= ϵ
 | "else" "{" <block> "}"

<switch> ::= "switch" "(" <expression> ")" "{" <case_blocks> <default_block_opt> "}"

<case_blocks> ::= <case_block>
| <case_blocks> <case_block>

<case_block> ::= "case" <expression> ":" "{" <block> "}"

<default_block_opt> ::= ϵ
| "default" ":" "{" <block> "}"

<print> ::= "out.print" "(" <STRING> ")"
| "out.print" "(" <STRING> "," <identifier_list> ")"

<return_stmt> ::= "return" <assigned_value> ";"
| "return" ";"

Flex

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

```
digit [0-9]  
letter [a-zA-Z]
```

```
%%  
"int" { return INT_DATA_TYPE; }  
"char" { return CHAR_DATA_TYPE; }  
"double" { return DOUBLE_DATA_TYPE; }  
"boolean" { return BOOLEAN_DATA_TYPE; }  
"void" { return VOID_DATA_TYPE; }  
"String" { return STRING_DATA_TYPE; }
```

```
"true" { return BOOLEAN_TRUE; }  
"false" { return BOOLEAN_FALSE; }
```

```
"public" { return PUBLIC; }  
"private" { return PRIVATE; }
```

```

"new" { return NEW; }
"class" { return CLASS; }
"return" { return RETURN; }
"break" { return BREAK; }
"do" { return DO; }
"while" { return WHILE; }
"for" { return FOR; }
"if" { return IF; }
"else" { return ELSE; }
"else if" { return ELSEIF; }
"switch" { return SWITCH; }
"case" { return CASE; }
"default" { return DEFAULT; }
"out.print" { return PRINT; }

"==" { return EQ_OP; }
"!=" { return NEQ_OP; }
">" { return GT_OP; }
"<" { return LT_OP; }
"&&" { return AND_OP; }
"||" { return OR_OP; }

";" { return SEMICOLON; }
"," { return COMMA; }
"(" { return LPAREN; }
")" { return RPAREN; }
"{" { return LBRACE; }
"}" { return RBRACE; }
"." { return DOT; }
"=" { return ASSIGN; }
"+" { return PLUS; }
"-" { return MINUS; }
"*" { return MULT; }
"/" { return DIV; }
":" { return COLON; }

[A-Z]({letter}|{digit}|_)* { yylval.str = strdup(yytext); return CLASS_IDENTIFIER; }
{letter}({letter}|{digit}|_)* { yylval.str = strdup(yytext); return IDENTIFIER; }

{digit}+ { yylval.intval = atoi(yytext); return INT; }
{digit}+"."{digit}+ { yylval.dblval = atof(yytext); return DOUBLE; }

```

```

\'\' {yyval.charval = yytext[0]; return CHAR;}
\"({letter}|{digit}|[ ~])*\\" { yyval.str = strdup(yytext); return STRING; }

"/".* { /* ignore comments */ }
"/*"(.|[ ~^*/]|(\n))*"/" { /* ignore comments */ }

[ \t\n]+ { /* ignore whitespace */ }

%%

int yywrap() {
    return 1;
}

```

Bison

```

%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
extern FILE *yyin;

extern int yylineno;
void yyerror(const char *s);
int yylex(void);

%}

#define parse.error verbose

%union {
    int intval;
    double dblval;
    char *str;
    char charval;
}

%token <intval> INT
%token <dblval> DOUBLE
%token <charval> CHAR
%token <str> IDENTIFIER CLASS_IDENTIFIER STRING

```

```

%token INT_DATA_TYPE CHAR_DATA_TYPE DOUBLE_DATA_TYPE
BOOLEAN_DATA_TYPE VOID_DATA_TYPE STRING_DATA_TYPE
%token BOOLEAN_TRUE BOOLEAN_FALSE
%token PUBLIC PRIVATE
%token NEW CLASS RETURN BREAK DO WHILE FOR IF ELSE ELSEIF SWITCH
CASE DEFAULT PRINT
%token COMMA SEMICOLON
%token LPAREN RPAREN LBRACE RBACE DOT ASSIGN PLUS MINUS MULT DIV
COLON
%token EQ_OP NEQ_OP GT_OP LT_OP AND_OP OR_OP

```

```

%left IDENTIFIER
%%

```

program:

```

    class_declarations { printf("-- Successful Parse :D --\n"); }
;

```

class_declarations:

```

    class_declaration
| class_declarations class_declaration
;

```

class_declaration:

```

    modifier CLASS CLASS_IDENTIFIER LBRACE class_body RBACE
;

```

modifier:

```

    PUBLIC
| PRIVATE
;

```

class_body:

```

    /* empty */
| class_body_elements
;

```

class_body_elements:

```

    class_body_element
| class_body_elements class_body_element
;

```

class_body_element:

```

    declaration

```

```
| method_declaration  
| class_declaration  
| assignment  
;
```

declaration:

```
data_type IDENTIFIER SEMICOLON  
| modifier data_type IDENTIFIER SEMICOLON  
| data_type IDENTIFIER ASSIGN assigned_value SEMICOLON  
| modifier data_type IDENTIFIER ASSIGN assigned_value SEMICOLON  
| data_type identifier_list SEMICOLON  
| data_type assignment_list SEMICOLON  
;
```

assignment_list:

```
IDENTIFIER ASSIGN assigned_value  
| assignment_list COMMA IDENTIFIER ASSIGN assigned_value  
;
```

data_type:

```
INT_DATA_TYPE  
| CHAR_DATA_TYPE  
| DOUBLE_DATA_TYPE  
| BOOLEAN_DATA_TYPE  
| VOID_DATA_TYPE  
| STRING_DATA_TYPE  
| CLASS_IDENTIFIER  
;
```

method_declaration:

```
modifier data_type IDENTIFIER LPAREN parameter_list RPAREN LBRACE block  
return_stmt RBRACE  
;
```

parameter_list:

```
/* empty */  
| parameters  
;
```

parameters:

```
parameter  
| parameters COMMA parameter
```

;

parameter:

data_type IDENTIFIER

;

method_call:

IDENTIFIER LPAREN identifier_list RPAREN SEMICOLON

identifier_list:

/* empty */

| identifiers

;

identifiers:

IDENTIFIER

| identifiers COMMA IDENTIFIER

| member_access

;

member_access:

IDENTIFIER DOT IDENTIFIER;

| IDENTIFIER DOT method_call

;

block:

statement

| block statement

;

statement:

declaration

| method_call

| assignment

| dowhile

| for

| if

| switch

| BREAK SEMICOLON

| print SEMICOLON

;

assignment:

IDENTIFIER ASSIGN assigned_value SEMICOLON

| IDENTIFIER ASSIGN NEW CLASS_IDENTIFIER LPAREN identifier_list RPAREN
SEMICOLON

| member_access ASSIGN assigned_value SEMICOLON
| member_access ASSIGN method_call
| IDENTIFIER ASSIGN method_call
;

assigned_value:

expression
| CHAR
| STRING
| BOOLEAN_TRUE
| BOOLEAN_FALSE
;

expression:

term
| expression MINUS term
| expression PLUS term
;

term:

factor
| term MULT factor
| term DIV factor
;

factor:

IDENTIFIER
| INT
| MINUS INT
| DOUBLE
| MINUS DOUBLE
| LPAREN expression RPAREN
| member_access
;

condition:

assigned_value
| assigned_value logic_operator assigned_value
;

logic_operator:

EQ_OP


```

| NEQ_OP
| GT_OP
| LT_OP
| AND_OP
| OR_OP
;

dowhile:
    DO LBRACE block RBACE WHILE LPAREN condition RPAREN SEMICOLON
;

for:
    FOR LPAREN assignment condition SEMICOLON assignment RPAREN LBRACE block
    RBACE
    | FOR LPAREN assignment condition SEMICOLON assignment RPAREN statement
;

if:
    IF LPAREN condition RPAREN LBRACE block RBACE elseif_opt else_opt
;

elseif_opt:
    /* empty */
    | elseif
;

elseif:
    ELSEIF LPAREN condition RPAREN LBRACE block RBACE elseif_opt
;

else_opt:
    /* empty */
    | ELSE LBRACE block RBACE
;

switch:
    SWITCH LPAREN expression RPAREN LBRACE case_blocks default_block_opt
    RBACE
;

case_blocks:
    case_block
    | case_blocks case_block
;

```

```

case_block:
    CASE expression COLON LBRACE block RBRACE
    ;

default_block_opt:
    /* empty */
    | DEFAULT COLON LBRACE block RBRACE
    ;

print:
    PRINT LPAREN STRING RPAREN
    | PRINT LPAREN STRING COMMA identifier_list RPAREN
    ;

return_stmt:
    RETURN assigned_value SEMICOLON
    | RETURN SEMICOLON
    ;

%%

void yyerror(const char *s) {
    fprintf(stderr, "Error: %s in line %d\n", s, yylineno);
}

int main(int argc, char** argv) {
    if (argc == 2) {
        yyin = fopen(argv[1], "r");
        if (!yyin) {
            perror(argv[1]);
            return 1;
        }

        //PRINT SOURCE CODE
        FILE* file_copy = fopen(argv[1], "r");
        char c = fgetc(file_copy);
        while (c != EOF)
        {
            printf ("%c", c);
            c = fgetc(file_copy);
        }
        fclose(file_copy);
    }
}

```

```

        yyparse();
    }
    else
        printf("Usage: %s <filename>\n", argv[0]);
    return 0;
}

```

Ερώτημα 3^ο

BNF

```

<CHAR> ::= "\" <letter> "\"
<STRING> ::= "\"" ( <letter> | <digit> | [ -~ ] )* "\""
<INT> ::= <digit>+
<DOUBLE> ::= <digit>+ "." <digit>+
<CLASS_IDENTIFIER> ::= [A-Z] ( <letter> | <digit> | "_" )*
<IDENTIFIER> ::= <letter> ( <letter> | <digit> | "_" )*

<digit> ::= "0" | ... | "9"
<letter> ::= "a" | ... | "z" | "A" | ... | "Z"

<program> ::= <class_declarations>

<class_declarations> ::= <class_declaration>
                        | <class_declarations> <class_declaration>

<class_declaration> ::= <modifier> "class" <CLASS_IDENTIFIER> "{" <class_body> "}"

<modifier> ::= "public"
              | "private"

<class_body> ::= ε
              | <class_body_elements>

<class_body_elements> ::= <class_body_element>
                        | <class_body_elements> <class_body_element>

<class_body_element> ::= <declaration>
                       | <method_declaration>
                       | <class_declaration>
                       | <assignment>

```

```

<declaration> ::= <data_type> <IDENTIFIER> ";"
                | <modifier> <data_type> <IDENTIFIER> ";"
                | <data_type> <IDENTIFIER> "=" <assigned_value> ";"
                | <modifier> <data_type> <IDENTIFIER> "=" <assigned_value> ";"
                | <data_type> <IDENTIFIER> "=" <expression> ";"
                | <modifier> <data_type> <IDENTIFIER> "=" <expression> ";"
                | <data_type> <identifier_list> ";"
                | <data_type> <assignment_list> ";"

```

```

<assignment_list> ::= <IDENTIFIER> "=" <assigned_value>
                    | <assignment_list> "," <IDENTIFIER> "=" <assigned_value>
                    | <IDENTIFIER> "=" <expression>
                    | <assignment_list> "," <IDENTIFIER> "=" <expression>

```

```

<data_type> ::= "int"
              | "char"
              | "double"
              | "boolean"
              | "void"
              | "String"
              | <CLASS_IDENTIFIER>

```

```

<method_declaration> ::= <modifier> <data_type> <IDENTIFIER> "(" <parameter_list> ")"
                        "{" <block> <return_stmt> "}"

```

```

<parameter_list> ::= ε
                  | <parameters>

```

```

<parameters> ::= <parameter>
                | <parameters> "," <parameter>

```

```

<parameter> ::= <data_type> <IDENTIFIER>

```

```

<method_call> ::= <IDENTIFIER> "(" <method_call_list> ")" ";"

```

```

<method_call_list> ::= "(" ")"
                    | <method_identifiers> ")"

```

```

<method_identifiers> ::= "(" <IDENTIFIER>
                       | "(" <member_access>
                       | <method_identifiers> "," <IDENTIFIER>
                       | <method_identifiers> "," <member_access>

```

```

<identifier_list> ::= ε

```

```

| <identifiers>

<identifiers> ::= <IDENTIFIER>
| <identifiers> "," <IDENTIFIER>
| <member_access>

<member_access> ::= <IDENTIFIER> "." <IDENTIFIER>
| <IDENTIFIER> "." <method_call>

<block> ::= <statement>
| <block> <statement>

<statement> ::= <declaration>
| <method_call>
| <assignment>
| <dowhile>
| <for>
| <if>
| <switch>
| "break" ";"
| <print> ";"

<assignment> ::= <IDENTIFIER> "=" <assigned_value> ";"
| <IDENTIFIER> "=" <expression> ";"
| <IDENTIFIER> "=" "new" <CLASS_IDENTIFIER> "(" <identifier_list> ")" ";"
| <member_access> "=" <assigned_value> ";"
| <member_access> "=" <method_call> ";"
| <IDENTIFIER> "=" <method_call>

<assigned_value> ::= <CHAR>
| <STRING>
| "true"
| "false"

<expression> ::= <term>
| <expression> "-" <term>
| <expression> "+" <term>

<term> ::= <factor>
| <term> "*" <factor>
| <term> "/" <factor>

<factor> ::= <IDENTIFIER>
| <INT>

```

- | "-" <INT>
- | <DOUBLE>
- | "-" <DOUBLE>
- | "(" <expression> ")"
- | <member_access>

<condition> ::= <assigned_value>
 | <condition> <logic_operator> <assigned_value>
 | <expression>
 | <condition> <logic_operator> <expression>

<logic_operator> ::= "=="
 | "!="
 | ">"
 | "<"
 | "&&"
 | "||"

<dowhile> ::= "do" "{" <block> "}" "while" "(" <condition> ")" ";"

<for> ::= "for" "(" <assignment> <condition> ";" <assignment> ")" "{" <block> "}"
 | "for" "(" <assignment> <condition> ";" <assignment> ")" <statement>

<if> ::= "if" "(" <condition> ")" "{" <block> "}" <elseif_opt> <else_opt>

<elseif_opt> ::= ε
 | <elseif>

<elseif> ::= "else if" "(" <condition> ")" "{" <block> "}" <elseif_opt>

<else_opt> ::= ε
 | "else" "{" <block> "}"

<switch> ::= "switch" "(" <expression> ")" "{" <case_blocks> <default_block_opt> "}"

<case_blocks> ::= <case_block>
 | <case_blocks> <case_block>

<case_block> ::= "case" <expression> ":" "{" <block> "}"

<default_block_opt> ::= ε
 | "default" ":" "{" <block> "}"

<print> ::= "out.print" "(" <STRING> ")"

```
| "out.print" "(" <STRING> "," <identifier_list> ")"
```

```
<return_stmt> ::= "return" <assigned_value> ";"  
| "return" ";"
```

Flex

```
%{  
#include <stdio.h>  
#include <stdlib.h>  
#include "parser.tab.h"  
%}  
%option yylineno  
  
digit [0-9]  
letter [a-zA-Z]  
  
%%  
"int" { return INT_DATA_TYPE; }  
"char" { return CHAR_DATA_TYPE; }  
"double" { return DOUBLE_DATA_TYPE; }  
"boolean" { return BOOLEAN_DATA_TYPE; }  
"void" { return VOID_DATA_TYPE; }  
"String" { return STRING_DATA_TYPE; }  
  
"true" { return BOOLEAN_TRUE; }  
"false" { return BOOLEAN_FALSE; }  
  
"public" { return PUBLIC; }  
"private" { return PRIVATE; }  
  
"new" { return NEW; }  
"class" { return CLASS; }  
"return" { return RETURN; }  
"break" { return BREAK; }  
"do" { return DO; }  
"while" { return WHILE; }  
"for" { return FOR; }  
"if" { return IF; }  
"else" { return ELSE; }  
"else if" { return ELSEIF; }  
"switch" { return SWITCH; }
```

```

"case" { return CASE; }
"default" { return DEFAULT; }
"out.print" { return PRINT; }

"==" { return EQ_OP; }
"!=" { return NEQ_OP; }
">" { return GT_OP; }
"<" { return LT_OP; }
"&&" { return AND_OP; }
"||" { return OR_OP; }

";" { return SEMICOLON; }
"," { return COMMA; }
"(" { return LPAREN; }
")" { return RPAREN; }
"{" { return LBRACE; }
"}" { return RBRACE; }
"." { return DOT; }
"=" { return ASSIGN; }
"+" { return PLUS; }
"-" { return MINUS; }
"*" { return MULT; }
"/" { return DIV; }
":" { return COLON; }

[A-Z]({letter}|{digit}|_)* { yylval.str = strdup(yytext); return CLASS_IDENTIFIER; }
{letter}({letter}|{digit}|_)* { yylval.str = strdup(yytext); return IDENTIFIER; }

{digit}+ { yylval.intval = atoi(yytext); return INT; }
{digit}+"."{digit}+ { yylval.dblval = atof(yytext); return DOUBLE; }

\.' { yylval.charval = yytext[0]; return CHAR; }
\"({letter}|{digit}|[ ~])*\" { yylval.str = strdup(yytext); return STRING; }

\"/\".* { /* ignore comments */ }
\"/\"(. [ ~^* / ] | (\\n))\"*\"/\" { /* ignore comments */ }

[ \\t\\n]+ { /* ignore whitespace */ }

%%

int yywrap() {
    return 1;
}

```



```
}
```

Bison

```
%{
```

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

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

```
#include <string.h>
```

```
#define MAX_STACK_SIZE 100
```

```
extern FILE *yyin;
```

```
extern int yylineno;
```

```
void yyerror(const char *s);
```

```
int yylex(void);
```

```
//=====
```

```
typedef struct identifier
```

```
{
```

```
    char* name;
```

```
    int block_level;
```

```
    double value;
```

```
} Identifier;
```

```
typedef struct node
```

```
{
```

```
    Identifier *data;
```

```
    struct node *next;
```

```
} Node;
```

```
Node *method_head=NULL;
```

```
Node *identifier_head=NULL;
```

```
int current_block=0;
```

```
//=====
```

```
// Function to create a new node
```

```
Node* createNode(char* name, int block_level);
```

```
// Function to insert a node at the end of the list
```

```
void insertNode(Node **head, char* name, int block_level);
```

```

// Function to print the linked list
void printList(Node *head);

// Function to free memory allocated for the linked list
void freeList(Node *head);

// Function that searches for an identifier and checks if it has been declared and is in the right
scope
void searchErrors(Node *head, char* name);

// Fucntion that searches and returns the value of an identifier
double searchIdentifier(Node *head, char* name);

// Function that performs an operation via the pop_operand function and gets the result to the
identifier struct
double getDataToIdentifier(Node *head, char* name);

// Function that deletes from the identifier list all identifiers that are out of scope
void scope_collapse(Node** head, int current_block);

//=====

double operand_stack[MAX_STACK_SIZE];
char operator_stack[MAX_STACK_SIZE];
int operand_top = -1;
int operator_top = -1;

//=====

void push_operand(double value);

void push_operator(char op);

double pop_operand();

char pop_operator();

double perform_operation(char op, double val1, double val2);

//=====

void printSource(const char* filename);

%}

```

```

%define parse.error verbose

%union {
    int intval;
    double dblval;
    char *str;
    char charval;
}

%token <intval> INT
%token <dblval> DOUBLE
%token <charval> CHAR
%token <str> IDENTIFIER CLASS_IDENTIFIER STRING

%token INT_DATA_TYPE CHAR_DATA_TYPE DOUBLE_DATA_TYPE
BOOLEAN_DATA_TYPE VOID_DATA_TYPE STRING_DATA_TYPE
%token BOOLEAN_TRUE BOOLEAN_FALSE
%token PUBLIC PRIVATE
%token NEW CLASS RETURN BREAK DO WHILE FOR IF ELSE ELSEIF SWITCH
CASE DEFAULT PRINT
%token COMMA SEMICOLON
%token LPAREN RPAREN LBRACE RBRACE DOT ASSIGN PLUS MINUS MULT DIV
COLON
%token EQ_OP NEQ_OP GT_OP LT_OP AND_OP OR_OP

%left IDENTIFIER
%%

program:
    class_declarations { printf("-- Successful Parse :D --\n"); }
    ;

class_declarations:
    class_declaration
    | class_declarations class_declaration
    ;

class_declaration:
    modifier CLASS CLASS_IDENTIFIER LBRACE {current_block++;} class_body
    RBRACE {scope_collapse(&method_head, current_block);
    scope_collapse(&identifier_head, current_block); current_block--;}
    ;

```

```

modifier:
    PUBLIC
    | PRIVATE
    ;

class_body:
    /* empty */
    | class_body_elements
    ;

class_body_elements:
    class_body_element
    | class_body_elements class_body_element
    ;

class_body_element:
    declaration
    | method_declaration
    | class_declaration
    | assignment
    ;

declaration:
    data_type IDENTIFIER SEMICOLON { insertNode(&identifier_head, $2, current_block);
}
    | modifier data_type IDENTIFIER SEMICOLON { insertNode(&identifier_head, $3,
current_block); }
    | data_type IDENTIFIER ASSIGN assigned_value SEMICOLON {
insertNode(&identifier_head, $2, current_block); }
    | modifier data_type IDENTIFIER ASSIGN assigned_value SEMICOLON {
insertNode(&identifier_head, $3, current_block); }
    | data_type IDENTIFIER ASSIGN expression SEMICOLON {
                                insertNode(&identifier_head, $2, current_block);
                                double result=getDataToIdentifier(identifier_head, $2);
                                printf("%s=%.2lf\n", $2, result);
                                }
    | modifier data_type IDENTIFIER ASSIGN expression SEMICOLON {
                                insertNode(&identifier_head, $3, current_block);
                                double result =
getDataToIdentifier(identifier_head, $3);
                                printf("%s=%.2lf\n", $3, result);
                                }
    | data_type identifier_list SEMICOLON

```

```
| data_type assignment_list SEMICOLON  
;
```

assignment_list:

```
IDENTIFIER ASSIGN assigned_value {insertNode(&identifier_head, $1, current_block);}
| assignment_list COMMA IDENTIFIER ASSIGN
assigned_value {insertNode(&identifier_head, $3, current_block);}
| IDENTIFIER ASSIGN expression {insertNode(&identifier_head, $1, current_block);
getDataToIdentifier(identifier_head, $1);}
| assignment_list COMMA IDENTIFIER ASSIGN
expression {insertNode(&identifier_head, $3, current_block); searchErrors(identifier_head,
$3); getDataToIdentifier(identifier_head, $3);}
;
```

data_type:

```
INT_DATA_TYPE
| CHAR_DATA_TYPE
| DOUBLE_DATA_TYPE
| BOOLEAN_DATA_TYPE
| VOID_DATA_TYPE
| STRING_DATA_TYPE
| CLASS_IDENTIFIER
;
```

method_declaration:

```
modifier data_type IDENTIFIER LPAREN parameter_list RPAREN LBRACE
{insertNode(&method_head, $3, current_block); current_block++;} block return_stmt
RBRACE {scope_collapse(&method_head, current_block);
scope_collapse(&identifier_head, current_block); current_block--;}
;
```

method_call:

```
IDENTIFIER method_call_list SEMICOLON { searchErrors(method_head, $1); }
;
```

method_call_list:

```
LPAREN RPAREN
| method_identifiers RPAREN
;
```

method_identifiers:

```
LPAREN IDENTIFIER {insertNode(&identifier_head, $2, current_block);}
| LPAREN member_access
```

```

    | method_identifiers COMMA IDENTIFIER {insertNode(&identifier_head, $3,
current_block);}
    | method_identifiers COMMA member_access
    ;

```

```

parameter_list:
    /* empty */
    | parameters
    ;

```

```

parameters:
    parameter
    | parameters COMMA parameter
    ;

```

```

parameter:
    data_type IDENTIFIER {insertNode(&identifier_head, $2, current_block);}
    ;

```

```

identifier_list:
    /* empty */
    | identifiers
    ;

```

```

identifiers:
    IDENTIFIER {insertNode(&identifier_head, $1, current_block);}
    | member_access
    | identifiers COMMA IDENTIFIER {insertNode(&identifier_head, $3, current_block);}
    | identifiers COMMA member_access
    ;

```

```

member_access:
    IDENTIFIER DOT IDENTIFIER { searchErrors(identifier_head, $1);
searchErrors(identifier_head, $3); }
    | IDENTIFIER DOT method_call
    ;

```

```

block:
    statement
    | block statement
    ;

```

```

statement:

```

```

declaration
| method_call
| assignment
| dowhile
| for
| if
| switch
| BREAK SEMICOLON
| print SEMICOLON
;

```

assignment:

```

IDENTIFIER ASSIGN assigned_value SEMICOLON {searchErrors(identifier_head, $1);}
| IDENTIFIER ASSIGN expression SEMICOLON {
    searchErrors(identifier_head, $1);
    double result=getDataToIdentifier(identifier_head, $1);
    printf("%s=%.2lf\n", $1, result);
}
| IDENTIFIER ASSIGN NEW CLASS_IDENTIFIER LPAREN identifier_list RPAREN
SEMICOLON {searchErrors(identifier_head, $1);}
| member_access ASSIGN expression SEMICOLON
| member_access ASSIGN method_call
| IDENTIFIER ASSIGN method_call {searchErrors(identifier_head, $1);}
;

```

assigned_value:

```

CHAR
| STRING
| BOOLEAN_TRUE
| BOOLEAN_FALSE
;

```

expression:

```

term
| expression MINUS term {push_operator('-');}
| expression PLUS term {push_operator('+');}
;

```

term:

```

factor
| term MULT factor {push_operator('*');}
| term DIV factor {push_operator('/');}
;

```

factor:

```
    IDENTIFIER {searchErrors(identifier_head, $1);  
push_operand(searchIdentifier(identifier_head, $1));}  
    | INT {push_operand($1);}  
    | MINUS INT {push_operand(-$2);}  
    | DOUBLE {push_operand($1);}  
    | MINUS DOUBLE {push_operand(-$2);}  
    | LPAREN expression RPAREN  
    | member_access  
    ;
```

condition:

```
    assigned_value  
    | condition logic_operator assigned_value  
    | expression  
    | condition logic_operator expression  
    ;
```

logic_operator:

```
    EQ_OP  
    | NEQ_OP  
    | GT_OP  
    | LT_OP  
    | AND_OP  
    | OR_OP  
    ;
```

dowhile:

```
    DO LBRACE {current_block++;} block RBRACE {scope_collapse(&identifier_head,  
current_block);current_block--;} WHILE LPAREN condition RPAREN SEMICOLON  
    ;
```

for:

```
    FOR LPAREN assignment condition SEMICOLON assignment RPAREN LBRACE  
{current_block++;} block RBRACE {scope_collapse(&identifier_head,  
current_block);current_block--;}  
    | FOR LPAREN assignment condition SEMICOLON assignment RPAREN statement  
    ;
```

if:

```
    IF LPAREN condition RPAREN LBRACE {current_block++;} block RBRACE  
{scope_collapse(&identifier_head, current_block);current_block--;} elseif_opt else_opt  
    ;
```



```

elseif_opt:
    /* empty */
    | elseif
    ;

elseif:
    ELSEIF LPAREN condition RPAREN LBRACE {current_block++;} block RBRACE
    {scope_collapse(&identifier_head, current_block);current_block--;} elseif_opt
    ;

else_opt:
    /* empty */
    | ELSE LBRACE {current_block++;} block RBRACE {scope_collapse(&identifier_head,
current_block);current_block--;}
    ;

switch:
    SWITCH LPAREN expression RPAREN LBRACE {current_block++;} case_blocks
    default_block_opt RBRACE {scope_collapse(&identifier_head,
current_block);current_block--;}
    ;

case_blocks:
    case_block
    | case_blocks case_block
    ;

case_block:
    CASE expression COLON LBRACE {current_block++;} block RBRACE
    {scope_collapse(&identifier_head, current_block);current_block--;}
    ;

default_block_opt:
    /* empty */
    | DEFAULT COLON LBRACE {current_block++;} block RBRACE
    {scope_collapse(&identifier_head, current_block);current_block--;}
    ;

print:
    PRINT LPAREN STRING RPAREN
    | PRINT LPAREN STRING COMMA identifier_list RPAREN
    ;

return_stmt:

```

```

    RETURN assigned_value SEMICOLON
| RETURN SEMICOLON
;

%%

void yyerror(const char *s) {
    fprintf(stderr, "Error: %s in line %d\n", s, yylineno);
}

int main(int argc, char** argv) {
    if (argc == 2) {
        yyin = fopen(argv[1], "r");
        if (!yyin) {
            perror(argv[1]);
            return 1;
        }

        current_block = 0;

        printSource(argv[1]);

        yyparse();

        freeList(identifier_head);
        freeList(method_head);
    }
    else
        printf("Usage: %s <filename>\n", argv[0]);

    return 0;
}

// Function to create a new node
Node* createNode(char* name, int block) {
    Node *newNode = (Node*)malloc(sizeof(Node));
    if (newNode == NULL) {
        fprintf(stderr, "Memory allocation failed\n");
        exit(EXIT_FAILURE);
    }
    newNode->data = (Identifier*)malloc(sizeof(Identifier));
    if (newNode->data == NULL) {
        fprintf(stderr, "Memory allocation failed\n");

```

```

        free(newNode);
        exit(EXIT_FAILURE);
    }
    newNode->data->name = strdup(name);
    newNode->data->block_level = block;
    newNode->next = NULL;
    return newNode;
}

// Function to insert a node at the end of the list
void insertNode(Node **head, char* name, int block_level) {
    Node *newNode = createNode(name, block_level);
    if (*head == NULL) {
        *head = newNode;
    } else {
        Node *temp = *head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
}

// Function to print the linked list
void printList(Node *head) {
    Node *current = head;
    while (current != NULL) {
        printf("Name: %s, Block Level: %d \n", current->data->name,
current->data->block_level);
        current = current->next;
    }
    printf("NULL\n");
}

// Function to free memory allocated for the linked list
void freeList(Node *head) {
    Node *current = head;
    while (current != NULL) {
        Node *temp = current;
        current = current->next;
        free(temp->data->name);
        free(temp->data);
        free(temp);
    }
}

```

```
}
```

```
void searchErrors(Node *head, char* name) {  
    Node *current = head;  
  
    while (current != NULL) {  
        if (strcmp(current->data->name, name) == 0) {  
            return;  
        }  
        current = current->next;  
    }  
    printf("Error: identifier \"%s\" in line %d not declared in this scope.\n", name, yylineno);  
    exit(EXIT_FAILURE);  
}
```

```
double searchIdentifier(Node *head, char* name)  
{  
    Node *current = head;  
    while (current != NULL) {  
        if (strcmp(current->data->name, name) == 0)  
        {  
            return current->data->value;  
        }  
        current = current->next;  
    }  
}
```

```
double getDataToIdentifier(Node *head, char* name)  
{  
    Node *current = head;  
    while (current != NULL) {  
        if (strcmp(current->data->name, name) == 0)  
        {  
            current->data->value = pop_operand();  
            return current->data->value;  
        }  
        current = current->next;  
    }  
}
```

```
void scope_collapse(Node** head, int current_block) {  
    Node* current = *head;  
    Node* prev = NULL;
```

```

while (current != NULL) {
    if (current->data->block_level == current_block) {
        Node* to_delete = current;
        current = current->next; // Move to the next node

        if (prev == NULL) {
            // If the node to delete is the head, update the head pointer
            *head = current;
        } else {
            // Otherwise, link the previous node to the next node
            prev->next = current;
        }

        // Free the node and its data
        free(to_delete->data->name); // Free the name string
        free(to_delete->data);      // Free the Identifier
        free(to_delete);           // Free the Node
    } else {
        // Move to the next node, updating prev
        prev = current;
        current = current->next;
    }
}

}

void push_operand(double value) {
    if (operand_top >= MAX_STACK_SIZE - 1) {
        fprintf(stderr, "Operand stack overflow\n");
        exit(EXIT_FAILURE);
    }
    operand_stack[++operand_top] = value;
}

void push_operator(char op) {
    operator_stack[++operator_top] = op;
    if (operator_top >= MAX_STACK_SIZE - 1) {
        fprintf(stderr, "Operator stack overflow\n");
        exit(EXIT_FAILURE);
    }
    while (operator_top >= 0 && (operator_stack[operator_top] == '*' ||
operator_stack[operator_top] == '/' || (op == '+' || op == '-')))
    {
        char opr = pop_operator();

```

```

        double val2 = pop_operand();
        double val1 = pop_operand();
        push_operand(perform_operation(opr, val1, val2));
    }

}

double pop_operand() {
    if (operand_top < 0) {
        fprintf(stderr, "Operand stack underflow\n");
        exit(EXIT_FAILURE);
    }
    return operand_stack[operand_top--];
}

char pop_operator() {
    if (operator_top < 0) {
        fprintf(stderr, "Operator stack underflow\n");
        exit(EXIT_FAILURE);
    }
    return operator_stack[operator_top--];
}

double perform_operation(char op, double val1, double val2) {
    switch (op) {
        case '+': return val1 + val2;
        case '-': return val1 - val2;
        case '*': return val1 * val2;
        case '/':
            if (val2 == 0) {
                fprintf(stderr, "Division by zero\n");
                exit(EXIT_FAILURE);
            }
            return val1 / val2;
        default:
            fprintf(stderr, "Unknown operator: %c\n", op);
            exit(EXIT_FAILURE);
    }
    return 0; // should never reach here
}

void printSource(const char* file_name){
    FILE* file= fopen(file_name, "r");

```

```

char c = fgetc(file);
printf("___Start of Source Code___\n");
while(c != EOF){
    printf("%c", c);
    c = fgetc(file);
}
fclose(file);
printf("\n___End of Source Code___\n");
}

```

Ερώτημα 4^ο

BNF

```

<CHAR> ::= "\" <letter> "\"
<STRING> ::= "'" ( <letter> | <digit> | [ ~ ] ) * "'"
<INT> ::= <digit> +
<DOUBLE> ::= <digit> + "." <digit> +
<CLASS_IDENTIFIER> ::= [A-Z] ( <letter> | <digit> | "_" ) *
<IDENTIFIER> ::= <letter> ( <letter> | <digit> | "_" ) *

<digit> ::= "0" | ... | "9"
<letter> ::= "a" | ... | "z" | "A" | ... | "Z"

<program> ::= <class_declarations>

<class_declarations> ::= <class_declaration>
                        | <class_declarations> <class_declaration>

<class_declaration> ::= <modifier> "class" <CLASS_IDENTIFIER> "{" <class_body> "}"

<modifier> ::= "public"
              | "private"

<class_body> ::= ε
              | <class_body_elements>

<class_body_elements> ::= <class_body_element>
                       | <class_body_elements> <class_body_element>

<class_body_element> ::= <declaration>
                       | <method_declaration>
                       | <class_declaration>

```

| <assignment>

```
<declaration> ::= <data_type> <IDENTIFIER> ";"  
    | <modifier> <data_type> <IDENTIFIER> ";"  
    | <data_type> <IDENTIFIER> "=" <assigned_value> ";"  
    | <modifier> <data_type> <IDENTIFIER> "=" <assigned_value> ";"  
    | <data_type> <IDENTIFIER> "=" <expression> ";"  
    | <modifier> <data_type> <IDENTIFIER> "=" <expression> ";"  
    | <data_type> <identifier_list> ";"  
    | <data_type> <assignment_list> ";"
```

```
<assignment_list> ::= <IDENTIFIER> "=" <assigned_value>  
    | <assignment_list> "," <IDENTIFIER> "=" <assigned_value>  
    | <IDENTIFIER> "=" <expression>  
    | <assignment_list> "," <IDENTIFIER> "=" <expression>
```

```
<data_type> ::= "int"  
    | "char"  
    | "double"  
    | "boolean"  
    | "void"  
    | "String"  
    | <CLASS_IDENTIFIER>
```

```
<method_declaration> ::= <modifier> <data_type> <IDENTIFIER> "(" <parameter_list> ")"  
    "{" <block> <return_stmt> "}"
```

```
<parameter_list> ::= ε  
    | <parameters>
```

```
<parameters> ::= <parameter>  
    | <parameters> "," <parameter>
```

```
<parameter> ::= <data_type> <IDENTIFIER>
```

```
<method_call> ::= <IDENTIFIER> "(" <method_call_list> ")" ";"
```

```
<method_call_list> ::= "(" "  
    | <method_identifiers> ")"
```

```
<method_identifiers> ::= "(" <IDENTIFIER>  
    | "(" <member_access>  
    | <method_identifiers> "," <IDENTIFIER>  
    | <method_identifiers> "," <member_access>
```


<identifier_list> ::= ϵ
| <identifiers>

<identifiers> ::= <IDENTIFIER>
| <identifiers> "," <IDENTIFIER>
| <member_access>

<member_access> ::= <IDENTIFIER> "." <IDENTIFIER>
| <IDENTIFIER> "." <method_call>

<block> ::= <statement>
| <block> <statement>

<statement> ::= <declaration>
| <method_call>
| <assignment>
| <dowhile>
| <for>
| <if>
| <switch>
| "break" ";"
| <print> ";"

<assignment> ::= <IDENTIFIER> "=" <assigned_value> ";"
| <IDENTIFIER> "=" <expression> ";"
| <IDENTIFIER> "=" "new" <CLASS_IDENTIFIER> "(" <identifier_list> ")" ";"
| <member_access> "=" <assigned_value> ";"
| <member_access> "=" <method_call> ";"
| <IDENTIFIER> "=" <method_call>

<assigned_value> ::= <CHAR>
| <STRING>
| "true"
| "false"

<expression> ::= <term>
| <expression> "-" <term>
| <expression> "+" <term>

<term> ::= <factor>
| <term> "*" <factor>
| <term> "/" <factor>

<factor> ::= <IDENTIFIER>
 | <INT>
 | "-" <INT>
 | <DOUBLE>
 | "-" <DOUBLE>
 | "(" <expression> ")"
 | <member_access>

<condition> ::= <assigned_value>
 | <condition> <logic_operator> <assigned_value>
 | <expression>
 | <condition> <logic_operator> <expression>

<logic_operator> ::= "=="
 | "!="
 | ">"
 | "<"
 | "&&"
 | "||"

<dowhile> ::= "do" "{" <block> "}" "while" "(" <condition> ")" ";"

<for> ::= "for" "(" <assignment> <condition> ";" <assignment> ")" "{" <block> "}"
 | "for" "(" <assignment> <condition> ";" <assignment> ")" <statement>

<if> ::= "if" "(" <condition> ")" "{" <block> "}" <elseif_opt> <else_opt>

<elseif_opt> ::= ϵ
 | <elseif>

<elseif> ::= "else if" "(" <condition> ")" "{" <block> "}" <elseif_opt>

<else_opt> ::= ϵ
 | "else" "{" <block> "}"

<switch> ::= "switch" "(" <expression> ")" "{" <case_blocks> <default_block_opt> "}"

<case_blocks> ::= <case_block>
 | <case_blocks> <case_block>

<case_block> ::= "case" <expression> ":" "{" <block> "}"

<default_block_opt> ::= ϵ
 | "default" ":" "{" <block> "}"

```
<print> ::= "out.print" "(" <STRING> ")"  
        | "out.print" "(" <STRING> ", " <identifier_list> ")"
```

```
<return_stmt> ::= "return" <assigned_value> ";"  
               | "return" ";"
```

Flex

```
%{  
#include <stdio.h>  
#include <stdlib.h>  
#include "parser.tab.h"  
%}  
%option yylineno  
  
digit [0-9]  
letter [a-zA-Z]  
  
%%  
"int" { return INT_DATA_TYPE; }  
"char" { return CHAR_DATA_TYPE; }  
"double" { return DOUBLE_DATA_TYPE; }  
"boolean" { return BOOLEAN_DATA_TYPE; }  
"void" { return VOID_DATA_TYPE; }  
"String" { return STRING_DATA_TYPE; }  
  
"true" { return BOOLEAN_TRUE; }  
"false" { return BOOLEAN_FALSE; }  
  
"public" { return PUBLIC; }  
"private" { return PRIVATE; }  
  
"new" { return NEW; }  
"class" { return CLASS; }  
"return" { return RETURN; }  
"break" { return BREAK; }  
"do" { return DO; }  
"while" { return WHILE; }  
"for" { return FOR; }  
"if" { return IF; }  
"else" { return ELSE; }
```

```

"else if" { return ELSEIF; }
"switch" { return SWITCH; }
"case" { return CASE; }
"default" { return DEFAULT; }
"out.print" { return PRINT; }

"==" { return EQ_OP; }
"!=" { return NEQ_OP; }
">" { return GT_OP; }
"<" { return LT_OP; }
"&&" { return AND_OP; }
"||" { return OR_OP; }

";" { return SEMICOLON; }
"," { return COMMA; }
"(" { return LPAREN; }
")" { return RPAREN; }
"{" { return LBRACE; }
"}" { return RBRACE; }
"." { return DOT; }
"=" { return ASSIGN; }
"+" { return PLUS; }
"-" { return MINUS; }
"*" { return MULT; }
"/" { return DIV; }
":" { return COLON; }

[A-Z]({letter}|{digit}|_)* { yylval.str = strdup(yytext); return CLASS_IDENTIFIER; }
{letter}({letter}|{digit}|_)* { yylval.str = strdup(yytext); return IDENTIFIER; }

{digit}+ { yylval.intval = atoi(yytext); return INT; }
{digit}+"."{digit}+ { yylval.dblval = atof(yytext); return DOUBLE; }

\.' {yylval.charval = yytext[0]; return CHAR;}
\"({letter}|{digit}|[ ~])*\" { yylval.str = strdup(yytext); return STRING; }

\"/\".* { /* ignore comments */ }
\"/*\"([ ~^*/]|(\n))*\"/*\" { /* ignore comments */ }

[ \t\n]+ { /* ignore whitespace */ }

%%

```

```
int yywrap() {
    return 1;
}
```

Bison

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_STACK_SIZE 100
extern FILE *yyin;

extern int yylineno;
void yyerror(const char *s);
int yylex(void);
//=====

typedef struct identifier
{
    char* name;
    int block_level;
    double value;
} Identifier;

typedef struct node
{
    Identifier *data;
    struct node *next;
} Node;

Node *method_head=NULL;
Node *identifier_head=NULL;

int current_block=0;

//=====

// Function to create a new node
Node* createNode(char* name, int block_level);

// Function to insert a node at the end of the list
```

```

void insertNode(Node **head, char* name, int block_level);

// Function to print the linked list
void printList(Node *head);

// Function to free memory allocated for the linked list
void freeList(Node *head);

// Function that searches for an identifier and checks if it has been declared and is in the right
scope
int searchErrors(Node *head, char* name);

// Fucntion that searches and returns the value of an identifier
double searchIdentifier(Node *head, char* name);

// Function that performs an operation via the pop_operand function and gets the result to the
identifier struct
double getDataToIdentifier(Node *head, char* name);

// Function that deletes from the identifier list all identifiers that are out of scope
void scope_collapse(Node** head, int current_block);

//=====

double operand_stack[MAX_STACK_SIZE];
char operator_stack[MAX_STACK_SIZE];
int operand_top = -1;
int operator_top = -1;

//=====

void push_operand(double value);

void push_operator(char op);

double pop_operand();

char pop_operator();

double perform_operation(char op, double val1, double val2);

//=====

void printSource(const char* filename);

```

```

%}

%define parse.error verbose

%union {
    int intval;
    double dblval;
    char *str;
    char charval;
}

%token <intval> INT
%token <dblval> DOUBLE
%token <charval> CHAR
%token <str> IDENTIFIER CLASS_IDENTIFIER STRING

%token INT_DATA_TYPE CHAR_DATA_TYPE DOUBLE_DATA_TYPE
BOOLEAN_DATA_TYPE VOID_DATA_TYPE STRING_DATA_TYPE
%token BOOLEAN_TRUE BOOLEAN_FALSE
%token PUBLIC PRIVATE
%token NEW CLASS RETURN BREAK DO WHILE FOR IF ELSE ELSEIF SWITCH
CASE DEFAULT PRINT
%token COMMA SEMICOLON
%token LPAREN RPAREN LBRACE RBRACE DOT ASSIGN PLUS MINUS MULT DIV
COLON
%token EQ_OP NEQ_OP GT_OP LT_OP AND_OP OR_OP

%left IDENTIFIER
%%

program:
    class_declarations { printf("-- Successful Parse :D --\n"); }
    ;

class_declarations:
    class_declaration
    | class_declarations class_declaration
    ;

class_declaration:
    modifier CLASS CLASS_IDENTIFIER LBRACE {current_block++;} class_body
    RBRACE { scope_collapse(&method_head, current_block);
scope_collapse(&identifier_head, current_block);current_block--;}

```

```

;

modifier:
    PUBLIC
    | PRIVATE
;

class_body:
    /* empty */
    | class_body_elements
;

class_body_elements:
    class_body_element
    | class_body_elements class_body_element
;

class_body_element:
    declaration
    | method_declaration
    | class_declaration
    | assignment
;

declaration:
    data_type IDENTIFIER SEMICOLON { insertNode(&identifier_head, $2, current_block);
}
    | modifier data_type IDENTIFIER SEMICOLON { insertNode(&identifier_head, $3,
current_block); }
    | data_type IDENTIFIER ASSIGN assigned_value SEMICOLON {
insertNode(&identifier_head, $2, current_block); }
    | modifier data_type IDENTIFIER ASSIGN assigned_value SEMICOLON {
insertNode(&identifier_head, $3, current_block); }
    | data_type IDENTIFIER ASSIGN expression SEMICOLON {
                                insertNode(&identifier_head, $2, current_block);
                                double result=getDataToIdentifier(identifier_head, $2);
                                printf("%s=%.2lf\n", $2, result);
                                }
    | modifier data_type IDENTIFIER ASSIGN expression SEMICOLON {
                                insertNode(&identifier_head, $3, current_block);
                                searchErrors(identifier_head, $3);
                                double result =
getDataToIdentifier(identifier_head, $3);

```



```

        printf("%s=%.2lf\n", $3, result);
    }
| data_type identifier_list SEMICOLON
| data_type assignment_list SEMICOLON
;

assignment_list:
    IDENTIFIER ASSIGN assigned_value {insertNode(&identifier_head, $1, current_block);}
    | assignment_list COMMA IDENTIFIER ASSIGN
    assigned_value {insertNode(&identifier_head, $3, current_block);}
    | IDENTIFIER ASSIGN expression {insertNode(&identifier_head, $1, current_block);
searchErrors(identifier_head, $1); getDataToIdentifier(identifier_head, $1);}
    | assignment_list COMMA IDENTIFIER ASSIGN
    expression {insertNode(&identifier_head, $3, current_block); searchErrors(identifier_head,
$3); getDataToIdentifier(identifier_head, $3);}
;

data_type:
    INT_DATA_TYPE
    | CHAR_DATA_TYPE
    | DOUBLE_DATA_TYPE
    | BOOLEAN_DATA_TYPE
    | VOID_DATA_TYPE
    | STRING_DATA_TYPE
    | CLASS_IDENTIFIER
;

method_declaration:
    modifier data_type IDENTIFIER LPAREN parameter_list RPAREN LBRACE
    {insertNode(&method_head, $3, current_block); current_block++;} block return_stmt
    RBRACE {scope_collapse(&method_head, current_block);
scope_collapse(&identifier_head, current_block);current_block--;}
;

method_call:
    IDENTIFIER method_call_list SEMICOLON { searchErrors(method_head, $1); }
;

method_call_list:
    LPAREN RPAREN
    | method_identifiers RPAREN
;

```

```

method_identifiers:
    LPAREN IDENTIFIER {insertNode(&identifier_head, $2, current_block);}
    | LPAREN member_access
    | method_identifiers COMMA IDENTIFIER {insertNode(&identifier_head, $3,
current_block);}
    | method_identifiers COMMA member_access
    ;

parameter_list:
    /* empty */
    | parameters
    ;

parameters:
    parameter
    | parameters COMMA parameter
    ;

parameter:
    data_type IDENTIFIER {insertNode(&identifier_head, $2, current_block);}
    ;

identifier_list:
    /* empty */
    | identifiers
    ;

identifiers:
    IDENTIFIER {insertNode(&identifier_head, $1, current_block);}
    | member_access
    | identifiers COMMA IDENTIFIER {insertNode(&identifier_head, $3, current_block);}
    | identifiers COMMA member_access
    ;

member_access:
    IDENTIFIER DOT IDENTIFIER { searchErrors(identifier_head, $1);
searchErrors(identifier_head, $3); }
    | IDENTIFIER DOT method_call
    ;

block:
    statement
    | block statement

```

;

statement:

declaration
| method_call
| assignment
| dowhile
| for
| if
| switch
| BREAK SEMICOLON
| print SEMICOLON
;

assignment:

IDENTIFIER ASSIGN assigned_value SEMICOLON {searchErrors(identifier_head, \$1);}
| IDENTIFIER ASSIGN expression SEMICOLON {
 if(!searchErrors(identifier_head, \$1)){
 double result=getDataToIdentifier(identifier_head, \$1);
 printf("%s=%.2lf\n", \$1, result);
 }
}
| IDENTIFIER ASSIGN NEW CLASS_IDENTIFIER LPAREN identifier_list RPAREN
SEMICOLON {searchErrors(identifier_head, \$1);}
| member_access ASSIGN expression SEMICOLON
| member_access ASSIGN method_call
| IDENTIFIER ASSIGN method_call {searchErrors(identifier_head, \$1);}
;

assigned_value:

CHAR
| STRING
| BOOLEAN_TRUE
| BOOLEAN_FALSE
;

expression:

term
| expression MINUS term {push_operator('-');}
| expression PLUS term {push_operator('+');}
;

term:

factor

```
| term MULT factor {push_operator('*');}
| term DIV factor {push_operator('/');}
;
```

factor:

```
IDENTIFIER {if(!searchErrors(identifier_head, $1))
push_operand(searchIdentifier(identifier_head, $1)); else push_operand(0);}
| INT {push_operand($1);}
| MINUS INT {push_operand(-$2);}
| DOUBLE {push_operand($1);}
| MINUS DOUBLE {push_operand(-$2);}
| LPAREN expression RPAREN
| member_access
;
```

condition:

```
assigned_value
| condition logic_operator assigned_value
| expression
| condition logic_operator expression
;
```

logic_operator:

```
EQ_OP
| NEQ_OP
| GT_OP
| LT_OP
| AND_OP
| OR_OP
;
```

dowhile:

```
DO LBRACE {current_block++;} block RBRACE {scope_collapse(&identifier_head,
current_block);current_block--;} WHILE LPAREN condition RPAREN SEMICOLON
;
```

for:

```
FOR LPAREN assignment condition SEMICOLON assignment RPAREN LBRACE
{current_block++;} block RBRACE {scope_collapse(&identifier_head,
current_block);current_block--;}
| FOR LPAREN assignment condition SEMICOLON assignment RPAREN statement
;
```

if:

```

    IF LPAREN condition RPAREN LBRACE {current_block++;} block RBRACE
    {scope_collapse(&identifier_head, current_block);current_block--;} elseif_opt else_opt
    ;

```

```

elseif_opt:
    /* empty */
    | elseif
    ;

```

```

elseif:
    ELSEIF LPAREN condition RPAREN LBRACE {current_block++;} block RBRACE
    {scope_collapse(&identifier_head, current_block);current_block--;} elseif_opt
    ;

```

```

else_opt:
    /* empty */
    | ELSE LBRACE {current_block++;} block RBRACE {scope_collapse(&identifier_head,
current_block);current_block--;}
    ;

```

```

switch:
    SWITCH LPAREN expression RPAREN LBRACE {current_block++;} case_blocks
    default_block_opt RBRACE {scope_collapse(&identifier_head,
current_block);current_block--;}
    ;

```

```

case_blocks:
    case_block
    | case_blocks case_block
    ;

```

```

case_block:
    CASE expression COLON LBRACE {current_block++;} block RBRACE
    {scope_collapse(&identifier_head, current_block);current_block--;}
    ;

```

```

default_block_opt:
    /* empty */
    | DEFAULT COLON LBRACE {current_block++;} block RBRACE
    {scope_collapse(&identifier_head, current_block);current_block--;}
    ;

```

```

print:
    PRINT LPAREN STRING RPAREN

```

```

| PRINT LPAREN STRING COMMA identifier_list RPAREN
;

return_stmt:
    RETURN assigned_value SEMICOLON
| RETURN SEMICOLON
;

%%

void yyerror(const char *s) {
    fprintf(stderr, "Error: %s in line %d\n", s, yylineno);
}

int main(int argc, char** argv) {
    if (argc == 2) {
        yyin = fopen(argv[1], "r");
        if (!yyin) {
            perror(argv[1]);
            return 1;
        }

        current_block = 0;

        printSource(argv[1]);

        yyparse();

        freeList(identifier_head);
        freeList(method_head);
    }
    else
        printf("Usage: %s <filename>\n", argv[0]);

    return 0;
}

// Function to create a new node
Node* createNode(char* name, int block) {
    Node *newNode = (Node*)malloc(sizeof(Node));
    if (newNode == NULL) {
        fprintf(stderr, "Memory allocation failed\n");
        exit(EXIT_FAILURE);
    }
}

```

```

    }
    newNode->data = (Identifier*)malloc(sizeof(Identifier));
    if (newNode->data == NULL) {
        fprintf(stderr, "Memory allocation failed\n");
        free(newNode);
        exit(EXIT_FAILURE);
    }
    newNode->data->name = strdup(name);
    newNode->data->block_level = block;
    newNode->next = NULL;
    return newNode;
}

// Function to insert a node at the end of the list
void insertNode(Node **head, char* name, int block_level) {
    Node *newNode = createNode(name, block_level);
    if (*head == NULL) {
        *head = newNode;
    } else {
        Node *temp = *head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
}

// Function to print the linked list
void printList(Node *head) {
    Node *current = head;
    while (current != NULL) {
        printf("Name: %s, Block Level: %d \n", current->data->name,
current->data->block_level);
        current = current->next;
    }
    printf("NULL\n");
}

// Function to free memory allocated for the linked list
void freeList(Node *head) {
    Node *current = head;
    while (current != NULL) {
        Node *temp = current;
        current = current->next;
    }
}

```

```

        free(temp->data->name);
        free(temp->data);
        free(temp);
    }
}

int searchErrors(Node *head, char* name) {
    Node *current = head;

    while (current != NULL) {
        if (strcmp(current->data->name, name) == 0) {
            return 0;
        }
        current = current->next;
    }
    printf("Error: identifier \"%s\" in line %d not declared in this scope.\n", name, yylineno);
    return 1;
}

double searchIdentifier(Node *head, char* name)
{
    Node *current = head;
    while (current != NULL) {
        if (strcmp(current->data->name, name) == 0)
        {
            return current->data->value;
        }
        current = current->next;
    }
}

double getDataToIdentifier(Node *head, char* name)
{
    Node *current = head;
    while (current != NULL) {
        if (strcmp(current->data->name, name) == 0)
        {
            current->data->value = pop_operand();
            return current->data->value;
        }
        current = current->next;
    }
}

```



```

void scope_collapse(Node** head, int current_block) {
    Node* current = *head;
    Node* prev = NULL;

    while (current != NULL) {
        if (current->data->block_level == current_block) {
            Node* to_delete = current;
            current = current->next; // Move to the next node

            if (prev == NULL) {
                // If the node to delete is the head, update the head pointer
                *head = current;
            } else {
                // Otherwise, link the previous node to the next node
                prev->next = current;
            }

            // Free the node and its data
            free(to_delete->data->name); // Free the name string
            free(to_delete->data);      // Free the Identifier
            free(to_delete);           // Free the Node
        } else {
            // Move to the next node, updating prev
            prev = current;
            current = current->next;
        }
    }
}

void push_operand(double value) {
    if (operand_top >= MAX_STACK_SIZE - 1) {
        fprintf(stderr, "Operand stack overflow\n");
        exit(EXIT_FAILURE);
    }
    operand_stack[++operand_top] = value;
}

void push_operator(char op) {
    operator_stack[++operator_top] = op;
    if (operator_top >= MAX_STACK_SIZE - 1) {
        fprintf(stderr, "Operator stack overflow\n");
        exit(EXIT_FAILURE);
    }
}

```

```

    while (operator_top >= 0 && (operator_stack[operator_top] == '*' ||
operator_stack[operator_top] == '/' || (op == '+' || op == '-')))
    {
        char opr = pop_operator();
        double val2 = pop_operand();
        double val1 = pop_operand();
        push_operand(perform_operation(opr, val1, val2));
    }

}

double pop_operand() {
    if (operand_top < 0) {
        fprintf(stderr, "Operand stack underflow\n");
        exit(EXIT_FAILURE);
    }
    return operand_stack[operand_top--];
}

char pop_operator() {
    if (operator_top < 0) {
        fprintf(stderr, "Operator stack underflow\n");
        exit(EXIT_FAILURE);
    }
    return operator_stack[operator_top--];
}

double perform_operation(char op, double val1, double val2) {
    switch (op) {
        case '+': return val1 + val2;
        case '-': return val1 - val2;
        case '*': return val1 * val2;
        case '/':
            if (val2 == 0) {
                fprintf(stderr, "Division by zero\n");
                exit(EXIT_FAILURE);
            }
            return val1 / val2;
        default:
            fprintf(stderr, "Unknown operator: %c\n", op);
            exit(EXIT_FAILURE);
    }
    return 0; // should never reach here
}

```

```
void printSource(const char* file_name){
    FILE* file= fopen(file_name, "r");
    char c = fgetc(file);
    printf("___Start of Source Code___\n");
    while(c != EOF){
        printf("%c", c);
        c = fgetc(file);
    }
    fclose(file);
    printf("\n___End of Source Code___\n");
}
```