Εργαστηριακή Άσκηση
Flex/Bison

ΑΡΧΕΣ ΓΛΩΣΣΩΝ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ ΚΑΙ ΜΕΤΑΦΡΑΣΤΩΝ

Αυγουστίνου Αγάπη, 1093327
Δούβρη Αγγελική, 1097441
Κατσαντάς Θεόδωρος, 1097459
Παγώνη Βασιλική, 1072497

Τμήμα Μηχανικών Η/Υ και Πληροφορικής,
Πανεπιστήμιο Πατρών

Ιούνιος 2024

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΜΕΡΟΣ Α: BNF GRAMMAR

1.1 Γραμματική

<program> ::= <include_statement>* <class_declaration>*

<include_statement> ::= "include" <string_literal> | "#" <string_literal>

<class_declaration> ::= "class" <identifier> "{" <class_body> "}"

<class_body> ::= (<field_declaration> | <method_declaration>)*

<field_declaration> ::= <type> <identifier> ";"

<method_declaration> ::= <type> <identifier> "(" <parameter_list>? ")" "{" <statement>* "}"

<parameter_list> ::= <parameter> ("," <parameter>)*

<parameter> ::= <type> <identifier>

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

<statement> ::= <variable_declaration> | <assignment> | <if_statement> |
<while_statement> | <for_statement> | <return_statement> |
<expression_statement> | <print_statement> | <block_statement>

<variable_declaration> ::= <type> <identifier> ("=" <expression>)? ";"

<assignment> ::= <identifier> "=" <expression> ";"

<if_statement> ::= "if" "(" <expression> ")" <statement> ("else" <statement>)?

<while_statement> ::= "while" "(" <expression> ")" <statement>

<for_statement> ::= "for" "(" <variable_declaration>? ";" <expression>? ";" <expression>? ")"
<statement>

<return_statement> ::= "return" <expression>? ";"

<expression_statement> ::= <expression> ";"

<print_statement> ::= "printf" "(" <string_literal> ("," <expression>)* ")" ";" |
"out.print" "(" <expression> ")" ";"

<block_statement> ::= "{" <statement>* "}"

<expression> ::= <identifier> | <literal> | <binary_expression> | <unary_expression> |
<method_call>

<binary_expression> ::= <expression> <operator> <expression>

<unary_expression> ::= <operator> <expression>

<method_call> ::= <identifier> "(" <argument_list>? ")"

<argument_list> ::= <expression> ("," <expression>)*

<literal> ::= <int_literal> | <double_literal> | <string_literal> | <char_literal> | "true" | "false"

<identifier> ::= /[a-zA-Z_][a-zA-Z0-9_]*/

<int_literal> ::= /[0-9]+/

<double_literal> ::= /[0-9]+\.[0-9]+d/

<string_literal> ::= /"[^"\n]*"/

<char_literal> ::= /'[^']'/

<reserved_keyword> ::= "int" | "char" | "double" | "boolean" | "String" | "class" | "new" |
"return" | "void" | "if" | "else" | "while" | "do" | "for" |
"switch" | "case" | "default" | "break" | "true" | "false" |
"public" | "private"

<operator> ::= "=" | "==" | "!=" | "<" | ">" | "+" | "-" | "*" | "/" |
"&&" | "||" | "!" | "&" | "|" | "^" | "%" | "<=" | ">="

<delimiter> ::= "{" | "}" | "[" | "]" | "(" | ")" | ";" | "," | "."

<print_statement> ::= "printf" | "out.print"

<include> ::= "include" | "#"

1.2 Τεκμηρίωση

Η BNF γραμματική που περιγράφουμε παραπάνω ορίζει την γραμματική μιας απλής (c-like) προγραμματιστικής γλώσσας. Η γραμματική αυτή καθορίζει την δομή αλλά και τους κανόνες σύνταξης που πρέπει να ακολουθούν τα προγράμματα για να ανήκουν σε αυτή την γλώσσα. Συγκεκριμένα για την δομή, έχουμε συμπεριλάβει κανόνες για δηλώσεις αρχείων, κλάσεων και μεθόδων, όπως και τους τύπους δεδομένων (π.χ. int, char, double, boolean, string, void).

Η γραμματική μας προφανώς και περιέχει κανόνες για δηλώσεις μεταβλητών, ελέγχου ροής (if, while, for), αλλά και αναθέσεις τιμών. Επίσης, περιγράφει τον τρόπο δηλώσεων και των μεθόδων με τις αντίστοιχες παραμέτρους και τύπους επιστροφής.

Η γραμματική μας επίσης ορίζει και τους συνδυασμούς εκφράσεων αριθμητικών και λογικών, καθώς και η κλήση των μεθόδων που περιέχονται αυτοί όπως και την εκτύπωση των δεδομένων των συμβολοσειρών ή χαρακτήρων.

Τέλος, οριζουμε την σύνταξη που πρέπει να ακολουθεί ένα πρόγραμμα που μεταγλωττίζεται, ελέγχοντας την ορθότητα του κώδικα κατα την διαδικασία της ανάλυσης του προγράμματος. Με αυτόν τον τρόπο αποφεύγονται τα σφάλματα και αυξάνεται η αξιοπιστία του μεταγλωττιστή μας.

ΜΕΡΟΣ Β: FLEX & BISON

2.1 Λεκτικός Αναλυτής (FLEX)

Στο δεύτερο μέρος μας ζητήθηκε να υλοποιήσουμε έναν λεξικό και συντακτικό αναλυτή, χρησιμοποιώντας τα προγράμματα Flex και Bison. Αυτός παίρνει ως είσοδο ένα αρχείο γραμμένο στη ψευδογλώσσα που περιγράψαμε στο Α' μέρος και ελέγχει σε ένα πέρασμα εάν το πρόγραμμα είναι συντακτικά ορθό. Αφού κληθεί το πρόγραμμα από τη γραμμή εντολών, επιστρέφει το ίδιο στην οθόνη με ένα διαγνωστικό μήνυμα που υποδεικνύει εάν είναι ορθά γραμμένο ή εάν υπάρχει σφάλμα. Στην περίπτωση σφάλματος η ανάλυση διακόπτεται στην γραμμή που αυτό παρουσιάζεται.

2.1.1 Κώδικας

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

extern char* current_token_text;
extern int yylineno;
extern int error_flag;
extern FILE* yyin;
int is_reserved(char* identifier) {
    return strcmp(identifier, "int") == 0 ||
           strcmp(identifier, "char") == 0 ||
           strcmp(identifier, "double") == 0 ||
           strcmp(identifier, "boolean") == 0 ||
           strcmp(identifier, "String") == 0 ||
           strcmp(identifier, "class") == 0 ||
           strcmp(identifier, "new") == 0 ||
           strcmp(identifier, "return") == 0 ||
           strcmp(identifier, "void") == 0 ||
           strcmp(identifier, "if") == 0 ||
           strcmp(identifier, "else") == 0 ||
           strcmp(identifier, "while") == 0 ||
           strcmp(identifier, "do") == 0 ||
           strcmp(identifier, "for") == 0 ||
           strcmp(identifier, "switch") == 0 ||
           strcmp(identifier, "case") == 0 ||
```

```
        strcmp(identifier, "default") == 0 ||
        strcmp(identifier, "break") == 0 ||
        strcmp(identifier, "true") == 0 ||
        strcmp(identifier, "false") == 0 ||
        strcmp(identifier, "public") == 0 ||
        strcmp(identifier, "private") == 0;
}


%}
%option yylineno

%%

\n                  { yylineno++; }
"include"             { current_token_text = strdup(yytext);  return INCLUDE; }
"#"                 { current_token_text = strdup(yytext);  return HASH; }
"int"               { current_token_text = strdup(yytext);  return INT_TYPE; }
"char"                { current_token_text = strdup(yytext);  return CHAR_TYPE; }
"double"              { current_token_text = strdup(yytext);  return DOUBLE_TYPE; }
"boolean"             { current_token_text = strdup(yytext);  return BOOLEAN_TYPE; }
"String"              { current_token_text = strdup(yytext);  return STRING_TYPE; }
"void"                { current_token_text = strdup(yytext);  return VOID_TYPE; }
"public"              { current_token_text = strdup(yytext);  return PUBLIC; }
"private"             { current_token_text = strdup(yytext);  return PRIVATE; }
"class"               { current_token_text = strdup(yytext);  return CLASS; }
"return"              { current_token_text = strdup(yytext);  return RETURN; }
"if"                { current_token_text = strdup(yytext);  return IF; }
"else"                { current_token_text = strdup(yytext);  return ELSE; }
"while"               { current_token_text = strdup(yytext);  return WHILE; }
"do"                { current_token_text = strdup(yytext);  return DO; }
"for"               { current_token_text = strdup(yytext);  return FOR; }
"switch"              { current_token_text = strdup(yytext);  return SWITCH; }
"case"                { current_token_text = strdup(yytext);  return CASE; }
"default"             { current_token_text = strdup(yytext);  return DEFAULT; }
"break"               { current_token_text = strdup(yytext);  return BREAK; }
"true"              { current_token_text = strdup(yytext);  return BOOLEAN_LITERAL; }
"false"               { current_token_text = strdup(yytext);  return BOOLEAN_LITERAL; }
"new"                 { current_token_text = strdup(yytext);  return NEW; }
"{"                 { current_token_text = strdup(yytext);  return OPEN_BRACE; }
"}"                 { current_token_text = strdup(yytext);  return CLOSE_BRACE; }
"["                 { current_token_text = strdup(yytext);  return OPEN_BRACKET; }
"]"                 { current_token_text = strdup(yytext);  return CLOSE_BRACKET; }
";"                 { current_token_text = strdup(yytext);  return SEMICOLON; }
"="                  { current_token_text = strdup(yytext);  return ASSIGN; }
"("                 { current_token_text = strdup(yytext);  return OPEN_PAREN; }
")"                 { current_token_text = strdup(yytext);  return CLOSE_PAREN; }
"'"                 { current_token_text = strdup(yytext);  return SINGLE_QUOTE; }
```

```
","                     { current_token_text = strdup(yytext);  return COMMA; }
\"                      { current_token_text = strdup(yytext);  return QUOTE; }
"out.print"               { current_token_text = strdup(yytext);  return PRINT; }
"<"                      { current_token_text = strdup(yytext);  return LESS_THAN; }
">"                      { current_token_text = strdup(yytext);  return GREATER_THAN; }
"!"                     { current_token_text = strdup(yytext);  return EXCLAMATION; }
"@"                       { current_token_text = strdup(yytext);  return AT; }
"$"                      { current_token_text = strdup(yytext);  return DOLLAR; }
"%"                       { current_token_text = strdup(yytext);  return PERCENT; }
"^"                      { current_token_text = strdup(yytext);  return CARET; }
"&"                      { current_token_text = strdup(yytext);  return AMPERSAND; }
"."                     { current_token_text = strdup(yytext);  return DOT; }
":"                     { current_token_text = strdup(yytext); return COLON; }
"+"                      { current_token_text = strdup(yytext); return PLUS; }
"-"                     { current_token_text = strdup(yytext); return MINUS; }
"*"                     { current_token_text = strdup(yytext); return MULTIPLY; }
"/"                     { current_token_text = strdup(yytext); return DIVIDE; }
"&&"                      { current_token_text = strdup(yytext); return AND; }
"||"                    { current_token_text = strdup(yytext); return OR; }
"=="                      { current_token_text = strdup(yytext); return EQUAL; }
"!="                     { current_token_text = strdup(yytext); return NOT_EQUAL; }

"printf"                  { current_token_text = strdup(yytext);  return PRINTF; }
[a-zA-Z_][a-zA-Z0-9_]*      {
                            current_token_text = strdup(yytext);
                            if (is_reserved(yytext)) {
                                printf("Error: '%s' is a reserved keyword and cannot be used as an
identifier.\n", yytext);
                                error_flag = 1;
                                return RESERVED_KEYWORD;
                            } else {
                                yylval.str = strdup(yytext);
                                return IDENTIFIER;
                            }
                        }
[0-9]+                    { current_token_text = strdup(yytext); yylval.str = strdup(yytext);  return
INTEGER_LITERAL; }
[0-9]+"."[0-9]+d            { current_token_text = strdup(yytext); yylval.str = strdup(yytext); return
DOUBLE_LITERAL; }
[ \t\n\r]+                ; // Ignore whitespace and newline characters
\/\/[^\n]*                ; // Single-line comment
\/\*([^*]|\*[^\/])*\*\/     ; // Multi-line comment
\"[^\"\n]*\"              { current_token_text = strdup(yytext); yylval.str = strdup(yytext); return
STRING_LITERAL; }
. { printf("Invalid token: %s at line %d\n", yytext, yylineno); exit(1); }
'.'                     { current_token_text = strdup(yytext); yylval.str = strdup(yytext); return
CHARACTER_LITERAL; }
```

```
%%

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

2.1.2 Τεκμηρίωση και παραδείγματα

Στο πρώτο μέρος του κώδικα flex, γραμμένο σε C, περιέχονται οι δηλώσεις. Εντός των συμβόλων%{ και %} εισάγαμε τις βιβλιοθήκες που σκοπεύαμε να χρησιμοποιήσουμε.Στην συνέχεια συμπεριλαμβάνουμε τις δηλώσεις των εξωτερικών μεταβλητών όπως φαίνεται και στην εικόνα 2.1.1.

```
extern char* current_token_text;
extern int yylineno;
extern int error_flag;
extern FILE* yyin;
```

Εικόνα 2.1.1

Αυτές οι εντολές δηλώνουν ότι οι μεταβλητές current_token_text, yylineno, error_flag και yyin είναι ορισμένες σε κάποιο άλλο αρχείο. Η λέξη-κλειδί extern χρησιμοποιείται για να δηλώσει ότι η μεταβλητή υπάρχει και είναι ορισμένη αλλού.

Η συνάρτηση is_reversed δέχεται ώς παράμετρο μία συμβολοσειρά και επιστρέφει έναν ακέραιο. Όπως φαίνεται στην εικόνα 2.1.2, η συνάρτηση strcmp μας συγκρίνει το identifier (αναγνωριστικό) με μια λίστα δεσμευμένων λέξεων (πχ. int,char,…κλπ.). Εάν το αναγνωριστικό ταιριάζει με κάποια από αυτές τις λέξεις τότε επιστρέφει 1 (true) αλλιώς εάν δεν ταιριάζουν επιστρέφει 0 (false).

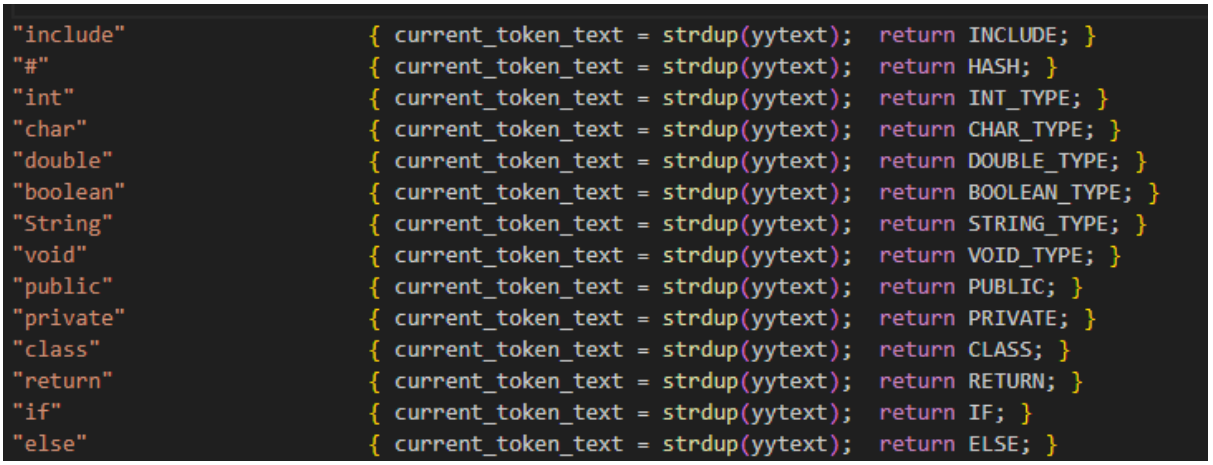
```
int is_reserved(char* identifier) {
    return strcmp(identifier, "int") == 0 ||
           strcmp(identifier, "char") == 0 ||
           strcmp(identifier, "double") == 0 ||
           strcmp(identifier, "boolean") == 0 ||
           strcmp(identifier, "String") == 0 ||
           strcmp(identifier, "class") == 0 ||
           strcmp(identifier, "new") == 0 ||
           strcmp(identifier, "return") == 0 ||
           strcmp(identifier, "void") == 0 ||
           strcmp(identifier, "if") == 0 ||
           strcmp(identifier, "else") == 0 ||
           strcmp(identifier, "while") == 0 ||
```

Εικόνα 2.1.2

Για να υλοποιήσουμε έναν compiler μας ενδιαφέρει να υπάρχει μία εντολή που θα κρατάει τον αριθμο γραμμής yylineno και ένας κανόνας που θα αυξάνει τον αριθμό γραμμής κάθε φορά που εντοπίζει ότι αλλάζει γραμμή το πρόγραμμα \n όπως φαίνεται στην εικόνα 2.1.3.

Εικόνα 2.1.3

Για κάθε keyword ή σύμβολο που αναγνωρίζουμε, η αντίστοιχη σειρά αντιγράφεται στην μεταβλητή current_token_text και επιστρέφει το αντιστοιχο token, όπως φαίνεται σε ένα δείγμα κώδικα στην εικόνα 2.1.4. Το token είναι ένας μοναδικός κωδικός για την επεξεργασία μιας λέξης.



Εικόνα 2.1.4

Στη συνέχεια, γράψαμε εντολές για να ορίσουμε τους ειδικούς χαρακτήρες ή συνδυασμούς αυτών, οι οποίες επιστρέφουν το συγκεκριμένο token που τους αντιστοιχεί όπως φαίνεται σε ένα μέρος του κώδικά μας στην εικόνα 2.1.5.

```
"!"                     { current_token_text = strdup(yytext);  return EXCLAMATION; }
"@"                     { current_token_text = strdup(yytext);  return AT; }
"$"                     { current_token_text = strdup(yytext);  return DOLLAR; }
"%"                     { current_token_text = strdup(yytext);  return PERCENT; }
"^"                     { current_token_text = strdup(yytext);  return CARET; }
"&"                     { current_token_text = strdup(yytext);  return AMPERSAND; }
"."                     { current_token_text = strdup(yytext);  return DOT; }
":"                     { current_token_text = strdup(yytext); return COLON; }
"+"                     { current_token_text = strdup(yytext); return PLUS; }
"-"                     { current_token_text = strdup(yytext); return MINUS; }
"*"                     { current_token_text = strdup(yytext); return MULTIPLY; }
"/"                     { current_token_text = strdup(yytext); return DIVIDE; }
"&&"                    { current_token_text = strdup(yytext); return AND; }
"||"                    { current_token_text = strdup(yytext); return OR; }
"=="                    { current_token_text = strdup(yytext); return EQUAL; }
```

Εικόνα 2.1.5

Στη συνέχεια, δημιουργήσαμε έναν κανόνα που αναγνωρίζει αλφαριθμητικά στοιχεία που αρχίζουν με γράμμα ή κάτω παύλα και ακολουθούνται από γράμματα, ψηφία ή άλλες κάτω παύλες. Αν το Identifier δεν είναι δεσμευμένη λέξη θα εκτυπωθεί ένα μήνυμα λάθους (επιστρέφει το token RESERVED_KEYWORD), αλλιώς επιστρέφεται το αναγνωριστικό ως το αντίστοιχο token IDENTIFIER (εικόνα 2.1.6).

```
"printf"                { current_token_text = strdup(yytext);  return PRINTF; }
[a-zA-Z_][a-zA-Z0-9_]*  {
                            current_token_text = strdup(yytext);
                            if (is_reserved(yytext)) {
                                printf("Error: '%s' is a reserved keyword and cannot be used as an i
                                error_flag = 1;
                                return RESERVED_KEYWORD;
                            } else {
                                yylval.str = strdup(yytext);
                                return IDENTIFIER;
                            }
```

Εικόνα 2.1.6

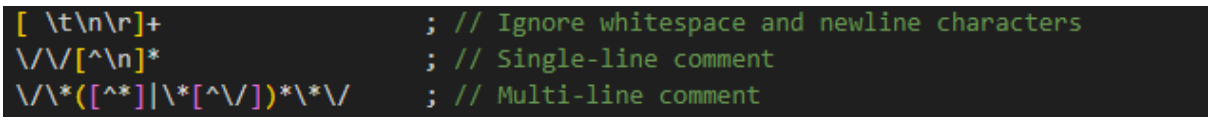
Στην εικόνα 2.1.7 δημιουργούμε κανόνες για την αναγνώριση ακολουθιών ψηφίων όπου τις επιστρέφει ώς ακέραιες σταθερές (INTEGER_LITERAL), και ακολουθίες ψηφίων που έχουν μία τελεία ανάμεσά τους και τις επιστρέφει ως σταθερές κινητής υποδιαστολής (DOUBLE_LITERAL).

```
                        }
[0-9]+                  { current_token_text = strdup(yytext); yylval.str = strdup(yytext);  return INTEGER_LITERAL; }
[0-9]+"."[0-9]+d        { current_token_text = strdup(yytext); yylval.str = strdup(yytext); return DOUBLE_LITERAL; }
```

Εικόνα 2.1.7

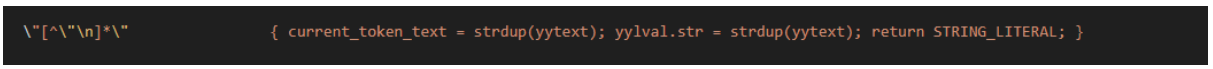
Οι παρακάτω κανόνες (εικόνα 2.1.8) κάνουν αντίστοιχα:

1) Αγνοεί τα κενά (tabs) και τους χαρακτήρες νέας γραμμής
2) Αγνοεί τα μονογραμμικά σχόλια που ξεκινούν με το σύμβολο // και εφαρμόζεται μέχρι το τέλος της γραμμής
3) Αγνοεί τα πολυγραμμικά σχόλια που ξεκινάνε με /* και τελειώνει με */



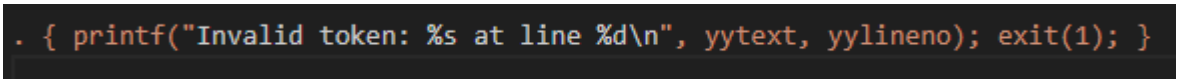
Εικόνα 2.1.8

Ο κανόνας αυτός, που φαίνεται και στην εικόνα 2.1.9, αναγνωρίζει σταθερές string που περικλείονται από διπλά εισαγωγικά χωρίς νέα γραμμή ανάμεσά τους.



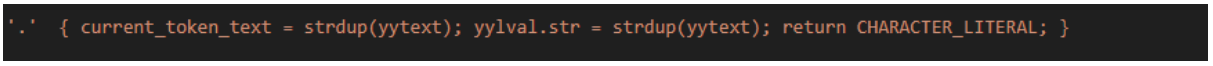
Εικόνες 2.1.9

Αυτός ο κανόνας, που φαίνεται και στην εικόνα 2.1.10, αναγνωρίζει όλους του υπόλοιπους χαρακτήρες οι οποίοι δεν ταιριάζουν με κανένα από τους προαναφέροντες κανόνες, και εκτυπώνει ένα μήνυμα λάθους και τερματίζει το πρόγραμμα.



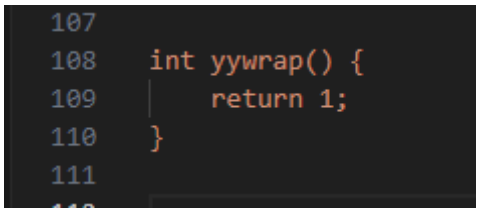
Εικόνα 2.1.10

Αυτός ο κανόνας, όπως φαίνεται και στην εικόνα 2.1.11, αναγνωρίζει σταθερές που περικλείονται από εισαγωγικά και τις επιστρέφει ως CHARACTER_LITERAL.



Εικόνα 2.1.11

Τέλος, στην εικόνα 2.1.12 βλέπουμε την συνάρτηση yywrap η οποία καλείται όταν φτάνουμε στο τέλος του αρχείου και προφανώς επιστρέφει την τιμή 1 για να δηλώσει στο Flex ότι η λεκτική ανάλυση έχει ολοκληρωθεί.



Εικόνα 2.1.12

Ο παραπάνω κώδικας Flex είναι χρήσιμος για την κατηγοριοποίηση διαφόρων στοιχείων (γραμμάτων,αριθμών, συμβόλων) σε μία γλώσσα προγραμματισμού παρέχοντας μας την βάση για την κατασκευή του μεταγλωττιστή μας.

2.2 Συντακτικός Αναλυτής (BISON)

2.2.1 Κώδικας

```
%union {
   int intval;
   char *str;
}

%left PLUS MINUS
%left MULTIPLY DIVIDE
%left OR
%left AND
%left EQUAL NOT_EQUAL
%left UNARY_MINUS
%nonassoc ASSIGN

%token <str> IDENTIFIER
%token <str> RESERVED_KEYWORD
%token <str> INTEGER_LITERAL
%token <str> CHARACTER_LITERAL
%token <str> DOUBLE_LITERAL
%token <str> BOOLEAN_LITERAL
%token <str> STRING_LITERAL
%token <str> INT_TYPE
%token <str> CHAR_TYPE
%token <str> DOUBLE_TYPE
%token <str> BOOLEAN_TYPE
%token <str> STRING_TYPE
%token <str> VOID_TYPE
%token PUBLIC PRIVATE CLASS OPEN_BRACE CLOSE_BRACE SEMICOLON
%token OPEN_PAREN CLOSE_PAREN PRINT HASH INCLUDE COMMA RETURN IF
ELSE DO WHILE FOR SWITCH CASE DEFAULT BREAK
%token LESS_THAN GREATER_THAN DOT EXCLAMATION AT DOLLAR PERCENT
CARET AMPERSAND PLUS MINUS DIVIDE MULTIPLY PRINTF
%token TRUE FALSE NEW COLON SINGLE_QUOTE
%token OPEN_BRACKET CLOSE_BRACKET QUOTE
%token <str> ASSIGN

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

int yylex();
void yyerror(const char *s);
extern FILE *yyin;
extern int error_flag;
```

```
extern int yylineno;
int assigned_value = 0;
char* assigned_string;
char assigned_char;
char* assigned_double;
char* assigned_boolean;

extern char *yytext;

char* current_token_text; // Token causing the error
char* methodName; // Global variable to store method name
char* var_type; // Variable to store type in variable_declaration
char* method_type; // Variable to store type in method_declaration
char* identifier_name; // To store identifier names
char* current_expression[1024];


typedef struct {
    char* name;
    char* value;
} VarAssignment;

VarAssignment var_list[100];
int var_list_size = 0;

typedef struct SymbolTableEntry {
    char* name;
    char* type;
    char* methodName;
    int value;
    struct SymbolTableEntry* next;
    double value_double;
    char* value_string;
    int value_boolean;
    char value_char;
} SymbolTableEntry;

SymbolTableEntry* variableTable = NULL;
SymbolTableEntry* methodTable = NULL;

typedef struct Scope {
    SymbolTableEntry* symbols;
    struct Scope* next;
} Scope;

Scope* scopeStack = NULL;

void pushScope() {
```

```
    Scope* newScope = (Scope*)malloc(sizeof(Scope));
    newScope->symbols = NULL;
    newScope->next = scopeStack;
    scopeStack = newScope;
}

void popScope() {
    if (scopeStack) {
        Scope* topScope = scopeStack;
        scopeStack = scopeStack->next;
        free(topScope);
    }
}

void addVariable(const char* name, int value) {
    SymbolTableEntry* entry = (SymbolTableEntry*)malloc(sizeof(SymbolTableEntry));
    if (entry == NULL) {
        fprintf(stderr, "Error: Memory allocation failed for variable '%s'\n", name);
        exit(EXIT_FAILURE);
    }

    entry->name = strdup(name);
    if (entry->name == NULL) {
        fprintf(stderr, "Error: Memory allocation failed for variable name '%s'\n", name);
        free(entry);
        exit(EXIT_FAILURE);
    }

    entry->value = value;
    entry->next = variableTable;
    variableTable = entry;
}


void addMethod(const char* name, const char* returnType) {
    SymbolTableEntry* entry = (SymbolTableEntry*)malloc(sizeof(SymbolTableEntry));
    entry->name = strdup(name);
    entry->type = strdup(returnType);
    entry->next = methodTable;
    methodTable = entry;
}

int checkVariableUsage(const char* name) {
    SymbolTableEntry* entry = variableTable;
    while (entry != NULL) {
        if (strcmp(entry->name, name) == 0) {
            printf("Variable %s found (line %d)\n", name, yylineno);
            return entry->value;
```

```
        }
        entry = entry->next;
    }
    printf("Error: Variable %s not defined (line %d)\n", name, yylineno);
    error_flag = 1;
    exit(EXIT_FAILURE);
    return 0; // This return statement is never reached, but added to avoid compiler warning
}

void checkMethodCall(const char* name) {
    SymbolTableEntry* entry = methodTable;
    while (entry != NULL) {
        if (strcmp(entry->name, name) == 0) {
            printf("Method %s found (line %d)\n", name, yylineno);
            return;
        }
        entry = entry->next;
    }
    printf("Error: Method %s not defined (line %d)\n", name, yylineno);
    error_flag = 1;
    exit(EXIT_FAILURE);
}


void setMethodName(char* name) {
    methodName = strdup(name);
}

void setVarType(char* type) {
    var_type = strdup(type);
}

void setMethodType(char* type) {
    method_type = strdup(type);
}

void setIdentifierName(char* name) {
    identifier_name = strdup(name);
}

void store_variable_value(const char* name, int value) {
    SymbolTableEntry* entry = variableTable;
    while (entry != NULL) {
        if (strcmp(entry->name, name) == 0) {
            // Print message indicating the assignment
            printf("Variable '%s' assigned value: %d\n", name, value);
            entry->value = value;
            return;
```

```
        }
        entry = entry->next;
    }

    // If the variable is not found, create a new entry
    SymbolTableEntry* newEntry = (SymbolTableEntry*)malloc(sizeof(SymbolTableEntry));
    newEntry->name = strdup(name);
    newEntry->value = value;
    newEntry->next = variableTable;
    variableTable = newEntry;

    // Print message for the new variable assignment
    printf("New variable '%s' assigned value: %d\n", identifier_name, value);
}

void store_variable_double_value(const char* name, double value) {
    SymbolTableEntry* entry = variableTable;
    while (entry != NULL) {
        if (strcmp(entry->name, name) == 0) {
            // Print message indicating the assignment
            printf("Variable '%s' assigned value: %lf\n", name, value);
            entry->value_double = value;
            return;
        }
        entry = entry->next;
    }

    // If the variable is not found, create a new entry
    SymbolTableEntry* newEntry = (SymbolTableEntry*)malloc(sizeof(SymbolTableEntry));
    newEntry->name = strdup(name);
    newEntry->value_double = value;
    newEntry->next = variableTable;
    variableTable = newEntry;

    // Print message for the new variable assignment
    printf("New variable '%s' assigned value: %lf\n", name, value);
}

void store_variable_string_value(const char* name, const char* value) {
    SymbolTableEntry* entry = variableTable;
    while (entry != NULL) {
        if (strcmp(entry->name, name) == 0) {
            // Print message indicating the assignment
            printf("Variable '%s' assigned value: %s\n", name, value);
            // Free the previously allocated value if it exists
            if (entry->value_string != NULL) {
                free(entry->value_string);
            }
```

```
            // Allocate memory and copy the new value
            entry->value_string = strdup(value);
            return;
        }
        entry = entry->next;
    }

    // If the variable is not found, create a new entry
    SymbolTableEntry* newEntry = (SymbolTableEntry*)malloc(sizeof(SymbolTableEntry));
    newEntry->name = strdup(name);
    newEntry->value = 0; // Assuming value is not used for strings
    newEntry->value_double = 0.0; // Assuming value_double is not used for strings
    newEntry->value_string = strdup(value);
    newEntry->next = variableTable;
    variableTable = newEntry;

    // Print message for the new variable assignment
    printf("New variable '%s' assigned value: %s\n", name, value);
}


void store_variable_boolean_value(const char* name, int value) {
    SymbolTableEntry* entry = variableTable;
    while (entry != NULL) {
        if (strcmp(entry->name, name) == 0) {
            // Print message indicating the assignment
            printf("Variable '%s' assigned value: %s\n", name, value ? "true" : "false");
            entry->value_boolean = value;
            return;
        }
        entry = entry->next;
    }

    // If the variable is not found, create a new entry
    SymbolTableEntry* newEntry = (SymbolTableEntry*)malloc(sizeof(SymbolTableEntry));
    newEntry->name = strdup(name);
    newEntry->value = 0; // Assuming value is not used for booleans
    newEntry->value_double = 0.0; // Assuming value_double is not used for booleans
    newEntry->value_string = NULL; // Assuming value_string is not used for booleans
    newEntry->value_boolean = value;
    newEntry->next = variableTable;
    variableTable = newEntry;

    // Print message for the new variable assignment
    printf("New variable '%s' assigned value: %s\n", name, value ? "true" : "false");
}

void store_variable_char_value(const char* name, char value) {
```

```
    SymbolTableEntry* entry = variableTable;
    while (entry != NULL) {
        if (strcmp(entry->name, name) == 0) {
            // Print message indicating the assignment
            printf("Variable '%s' assigned value: %c\n", name, value);
            entry->value_char = value;
            return;
        }
        entry = entry->next;
    }

    // If the variable is not found, create a new entry
    SymbolTableEntry* newEntry = (SymbolTableEntry*)malloc(sizeof(SymbolTableEntry));
    newEntry->name = strdup(name);
    newEntry->value = 0; // Assuming value is not used for chars
    newEntry->value_double = 0.0; // Assuming value_double is not used for chars
    newEntry->value_string = NULL; // Assuming value_string is not used for chars
    newEntry->value_char = value;
    newEntry->next = variableTable;
    variableTable = newEntry;

    // Print message for the new variable assignment
    printf("New variable '%s' assigned value: %c\n", name, value);
}

%}


%{
int error_flag = 0;
%}

%%

program:
        | program include_directive
        | program class
        | program method_declaration
        | program variable_declaration
        | program assignment_statement
        | program method_call
        ;

include_directive:
    HASH INCLUDE LESS_THAN IDENTIFIER DOT IDENTIFIER GREATER_THAN
    ;

class:
```

```
    opt_modifier CLASS identifier_cap OPEN_BRACE class_body CLOSE_BRACE
    ;

identifier_cap:
    IDENTIFIER {
        if (!isupper(yytext[0])) {
            printf("Error: Class name '%s' must start with a capital letter.\n", yytext);
            error_flag = 1;
        } else {
            setIdentifierName(yytext); // Set the identifier name
        }
    }
    ;

type:
    INT_TYPE { setVarType("int"); }
    | CHAR_TYPE { setVarType("char"); }
    | DOUBLE_TYPE { setVarType("double"); }
    | STRING_TYPE { setVarType("String"); }
    | VOID_TYPE { setVarType("void"); }
    | BOOLEAN_TYPE { setVarType("boolean");}
    ;

class_body:
    | class_body class_member
    | class_body class
    ;

class_member:
    variable_declaration
    | method_declaration
    | SEMICOLON
    ;

method_call:
    IDENTIFIER { setMethodName($1); } OPEN_PAREN argument_list CLOSE_PAREN
SEMICOLON{
        checkMethodCall(methodName);
    }
    ;

argument_list:
    | arguments
    ;

arguments:
    expression
    | arguments COMMA argument
```

```
    | type IDENTIFIER
    ;

argument:
    expression
    | type IDENTIFIER
    ;

statements:
    | statements statement
    ;

statement:
    variable_declaration
    | RETURN SEMICOLON
    | RETURN expression SEMICOLON
    | method_call SEMICOLON { checkMethodCall(methodName); }
    | variable_declaration SEMICOLON
    | assignment_statement
    | object_creation
    | method_declaration
    | print_statement
    | break_statement
    | do_while_statement
    | for_statement
    | switch_statement
    | if_statement
    ;

switch_statement:
    SWITCH OPEN_PAREN expression CLOSE_PAREN OPEN_BRACE case_clauses
default_clause_opt CLOSE_BRACE
    ;

case_clauses:
    case_clause
    | case_clauses case_clause
    ;

case_clause:
    CASE expression COLON statements
    ;

default_clause_opt:
    | DEFAULT COLON statements
    ;

if_statement:
```

```
    IF OPEN_PAREN ifexpression CLOSE_PAREN OPEN_BRACE statements
CLOSE_BRACE optional_else_if optional_else
    ;

optional_else_if:
    | optional_else_if ELSE IF OPEN_PAREN logical_expression CLOSE_PAREN
OPEN_BRACE statements CLOSE_BRACE
    ;

optional_else:
    | ELSE OPEN_BRACE statements CLOSE_BRACE
    ;

do_while_statement:
    DO OPEN_BRACE statements CLOSE_BRACE WHILE OPEN_PAREN
logical_expression CLOSE_PAREN SEMICOLON
    ;

print_statement:
    PRINT OPEN_PAREN STRING_LITERAL COMMA expression CLOSE_PAREN
SEMICOLON
    | PRINT OPEN_PAREN STRING_LITERAL CLOSE_PAREN SEMICOLON
    | PRINT OPEN_PAREN IDENTIFIER {setIdentifierName($3);}  CLOSE_PAREN
SEMICOLON {checkVariableUsage(identifier_name);}
    ;

break_statement:
    BREAK SEMICOLON
    ;

for_statement:
    FOR OPEN_PAREN for_init SEMICOLON for_condition SEMICOLON for_update
CLOSE_PAREN OPEN_BRACE statements CLOSE_BRACE
    ;

for_init:
    variable_declaration
    | assignment_statement
    ;

for_condition:
    logical_expression
    ;

for_update:
    IDENTIFIER ASSIGN expression
    ;
```

```
assignment_statement: IDENTIFIER {
        setIdentifierName($1);
    } DOT IDENTIFIER {
        setMethodName(yytext);
    } ASSIGN expression SEMICOLON {
        checkMethodCall(identifier_name);
    }
    | IDENTIFIER ASSIGN expression SEMICOLON {
            setIdentifierName($1);
            addVariable(identifier_name, assigned_value);
    }
    ;

object_creation:
    IDENTIFIER IDENTIFIER ASSIGN NEW IDENTIFIER OPEN_PAREN CLOSE_PAREN
SEMICOLON
    ;

variable_declaration:
    opt_modifier type IDENTIFIER {
        setIdentifierName($3);
    } opt_assignment SEMICOLON {
        // Add the variable to the symbol table
        addVariable(identifier_name, assigned_value);
        checkVariableUsage(identifier_name);
    }
    ;

opt_assignment:
    | ASSIGN INTEGER_LITERAL{
        assigned_value = atoi($2);
        // Print the assigned value
        if (assigned_value != -1) {
            printf("Variable '%s' assigned value: %d\n", identifier_name, assigned_value);
            assigned_value = -1; // Reset assigned value for next variable
        } else {
            // If no assignment, print a message indicating it
            printf("Variable '%s' declared without initialization\n", identifier_name);
        }
        } opt_assignment
    | ASSIGN DOUBLE_LITERAL {
        double assigned_double;
        sscanf($2, "%lf", &assigned_double); // Extract numeric part and convert to double
        store_variable_double_value($1, assigned_double); // Store the double value
        } opt_assignment
    | ASSIGN STRING_LITERAL {
        assigned_string = strdup($2);
        store_variable_string_value($1, assigned_string);
```

```
        }  opt_assignment

    | ASSIGN BOOLEAN_LITERAL  {
        if (strcmp($2, "true") == 0) {
            assigned_value = 0;
        } else {
            assigned_value = 1;
        }
        store_variable_boolean_value($1, assigned_value);
    }  opt_assignment
    | ASSIGN CHARACTER_LITERAL {
        assigned_char = $2[1];
        store_variable_char_value($1, assigned_char);

    } opt_assignment
    | COMMA opt_identifier opt_assignment
    | ASSIGN expression
    ;

opt_identifier:
              | IDENTIFIER {setIdentifierName($1);}
              ;

method_declaration:
    opt_modifier type IDENTIFIER { setIdentifierName($3); setMethodType(var_type); }
OPEN_PAREN argument_list CLOSE_PAREN  OPEN_BRACE statements CLOSE_BRACE
{
        addMethod(identifier_name, method_type);
        checkMethodCall(identifier_name);
    }
    ;

opt_modifier:
    | PUBLIC
    | PRIVATE
    ;

expression:
    term
    | expression PLUS term
    | expression MINUS term
    | class_member_access
    | IDENTIFIER ASSIGN expression
    ;

term:
    IDENTIFIER {checkVariableUsage($1);}
    | term MULTIPLY IDENTIFIER
```

```
    | term DIVIDE IDENTIFIER
    | MINUS IDENTIFIER %prec UNARY_MINUS
    | OPEN_PAREN expression CLOSE_PAREN
    ;



ifexpression:
    logical_expression
    | IDENTIFIER
    ;

logical_expression:
    logical_expression OR logical_term
    | logical_term
    ;

logical_term:
    logical_term AND logical_factor
    | logical_factor
    ;

logical_factor:
       expression EQUAL expression
    | expression NOT_EQUAL expression
    | expression LESS_THAN expression
    | expression GREATER_THAN expression
    | '(' logical_expression ')'
    ;

class_member_access:
    IDENTIFIER DOT IDENTIFIER
    ;

%%

void yyerror(const char *s) {
    printf("Syntax error at line %d: %s\n", yylineno, s);
    printf("Token encountered: %s\n", current_token_text);

    error_flag = 1;
    exit(EXIT_FAILURE); // Exit the program
}



int main(int argc, char *argv[]) {
    if (argc != 2) {
```

```
        printf("Usage: %s file_name\n", argv[0]);
        return 1;
    }

    FILE *fp = fopen(argv[1], "r");
    if (!fp) {
        printf("Error opening file %s\n", argv[1]);
        return 1;
    }

    yyin = fp;

    yyparse();

    fclose(fp);

    if (error_flag) {
        printf("Compilation failed with errors.\n");
        return 1;
    } else {
        printf("Compilation complete with no errors.\n");
        return 0;
    }
    return 0;
}
```

2.1.2 Τεκμηρίωση και παραδείγματα

Θα εξηγήσουμε τα σημαντικότερα σημεία του κώδικα που χρήζουν ανάλυσης για να μην πλατιάζουμε, οπότε θα αφήσουμε εκτός αναφοράς τις λεπτομέρειες για τον ορισμό των tokens,strings etc. και θα αναλύσουμε τις κλάσεις και τις μεθόδους που χρησιμοποιήσαμε.

Ορισμοί Δομών και Μεταβλητών

1) VarAssignment & λίστα μεταβλητών

Στην εικόνα 2.2.1 έχουμε μια δομή που αποθηκεύει μια μεταβλητή και την τιμή της, η οποία έχει δύο μέλη, το name και value, τα οποία αντιστοιχούν στο όνομα και την τιμή της μεταβλητής.

Στη συνέχεια, θα φτιάξουμε μια λίστα (var_list) που μπορεί να αποθηκεύσει μέχρι 100 δομές και θα ορίσουμε var_list_size ώστε να μετράει το μέγεθος της λίστας των μεταβλητών.

```
typedef struct {
    char* name;
    char* value;
} VarAssignment;

VarAssignment var_list[100];
int var_list_size = 0;
```

Εικόνα 2.2.1

2) SymbolTableEntry

Η δομή SymbolTableEntry (εικόνα 2.2.2) αποθηκεύει πληροφορίες σχετικά με μια συμβολική μεταβλητή και έχει διάφορα μέλη για τύπους και τιμές πληροφοριών

πχ. value_boolean -> αν η τιμή της μεταβλητής είναι boolean.

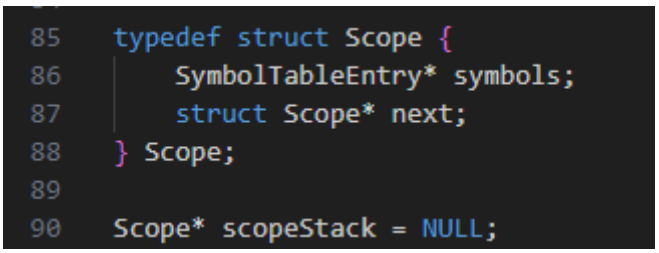
Στο τέλος, ορίζουμε τους πίνακες των μεθόδων και μεταβλητών, methodTable και variableTable αντίστοιχα, οι οποίοι είναι δείκτες στην αρχή της λίστας τους.

```
70  typedef struct SymbolTableEntry {
71      char* name;
72      char* type;
73      char* methodName;
74      int value;
75      struct SymbolTableEntry* next;
76      double value_double;
77      char* value_string;
78      int value_boolean;
79      char value_char;
80  } SymbolTableEntry;
81
82  SymbolTableEntry* variableTable = NULL;
83  SymbolTableEntry* methodTable = NULL;
```

Εικόνα 2.2.2

3) Scope & στοίβα πεδίων

Το scope, το οποίο φαίνεται στην εικόνα 2.2.3 είναι μια δομή η οποία αντιπροσωπεύει ένα πεδίο, scope, το οποίο περιέχει τις συμβολικές μεταβλητές. Αυτό που μας ενδιαφέρει εδώ είναι ότι έχουμε έναν δείκτη στην αρχή της λίστας των συμβόλων (symbols), έναν δείκτη στο επόμενο πεδίο (next) και έναν δείκτη στην κορυφή της στοίβας των πεδίων (scopeStack).



Εικόνα 2.2.3

Συναρτήσεις Διαχείρισης Στοίβας Πεδίων

1) Συνάρτηση pushScope

Η συνάρτηση pushScope (εικόνα 2.2.4) δημιουργεί ένα πεδίο στην κορυφή της στοίβας, το οποίο έχει άδεια λίστα συμβολοσειρών και δείχνει την προηγούμενη θέση της στοίβας.

```
void pushScope() {
    Scope* newScope = (Scope*)malloc(sizeof(Scope));
    newScope->symbols = NULL;
    newScope->next = scopeStack;
    scopeStack = newScope;
}
```

Εικόνα 2.2.4

2) Συνάρτηση popScope

Η συνάρτηση popScore (εικόνα 2.2.5) αφαιρεί το πεδίο στην κορυφή της στοίβας και αν η στοίβα δεν είναι άδεια ενημερώνει την κορυφή της στοίβας.

```
void popScope() {
    if (scopeStack) {
        Scope* topScope = scopeStack;
        scopeStack = scopeStack->next;
        free(topScope);
    }
}
```

Εικόνα 2.2.5

3) Συνάρτηση addVariable

Η συνάρτηση addVariable (εικόνα 2.2.6) χρησιμοποιεί την εντολή malloc για να δεσμεύσει την μνήμη (για το νέο αντικείμενο SymbolTableEntry), αν αποτύχει εμφανίζει μήνυμα αποτυχίας. Χρησιμοποιεί την strdup για να αντιγράψει το όνομα της μεταβλητής (πάλι έχουμε έλεγχο για αποτυχία οπου το πρόγραμμα τερματίζεται). Τέλος, αναθέτει ακέραια τιμή στην νέα μεταβλητή και την συνδέει με την αρχή της λίστας μεταβλητών.

```
void addVariable(const char* name, int value) {
    SymbolTableEntry* entry = (SymbolTableEntry*)malloc(sizeof(SymbolTableEntry));
    if (entry == NULL) {
        fprintf(stderr, "Error: Memory allocation failed for variable '%s'\n", name);
        exit(EXIT_FAILURE);
    }

    entry->name = strdup(name);
    if (entry->name == NULL) {
        fprintf(stderr, "Error: Memory allocation failed for variable name '%s'\n", name);
        free(entry);
        exit(EXIT_FAILURE);
    }

    entry->value = value;
    entry->next = variableTable;
    variableTable = entry;
}
```

Εικόνα 2.2.6

4) Συνάρτηση addMethod

Η συνάρτηση addMethod (εικόνα 2.2.7) δεσμεύει μνήμη για ένα νέο αντικείμενο SymbolTableEntry. Στην συνέχεια αντιγράφει την μέθοδο χρησιμοποιώντας την strdup. Τέλος, προσθέτει την νέα μέθοδο στην κορυφή του πίνακα μεθόδων.

```
void addMethod(const char* name, const char* returnType) {
    SymbolTableEntry* entry = (SymbolTableEntry*)malloc(sizeof(SymbolTableEntry));
    entry->name = strdup(name);
    entry->type = strdup(returnType);
    entry->next = methodTable;
    methodTable = entry;
}
```

Εικόνα 2.2.7

5) Συνάρτηση checkVariableUsage

Την συνάρτηση (εικόνα 2.2.8) την χρησιμοποιούμε για να ελέγξουμε αν μία μεταβλητή έχει χρησιμοποιηθεί. Στην αρχή η συνάρτηση ξεκινά από την αρχή της λίστας μεταβλητών και ελέγχει με την σειρά τα ονόματα και τα συγκρίνει με το δεδομένο όνομα. Εάν βρεθεί μεταβλητή εμφανίζει μήνυμα ότι βρέθηκε και επιστρέφει την τιμή της, ενώ αν δεν βρεθεί μεταβλητή εμφανίζει μήνυμα λάθους και το πρόγραμμα τερματίζει (error_flag).

```c
int checkVariableUsage(const char* name) {
    SymbolTableEntry* entry = variableTable;
    while (entry != NULL) {
        if (strcmp(entry->name, name) == 0) {
            printf("Variable %s found (line %d)\n", name, yylineno);
            return entry->value;
        }
        entry = entry->next;
    }
    printf("Error: Variable %s not defined (line %d)\n", name, yylineno);
    error_flag = 1;
    exit(EXIT_FAILURE);
    return 0; // This return statement is never reached, but added to avoid compiler warning
}
```

Εικόνα 2.2.8

6) Συνάρτηση checkMethodCall

Η συνάρτηση checkMethodCall (εικόνα 2.2.9) αναζητά την μέθοδο με το όνομα που δώσαμε με το όνομα στην λίστα μεθόδων. Εάν βρεθεί εμφανίζει μήνυμα ότι η μέθοδος βρέθηκε ενώ αν δεν βρεθεί εμφανίζει μήνυμα λάθους και error_flag =1; και τερματίζει το πρόγραμμα.

```c
void checkMethodCall(const char* name) {
    SymbolTableEntry* entry = methodTable;
    while (entry != NULL) {
        if (strcmp(entry->name, name) == 0) {
            printf("Method %s found (line %d)\n", name, yylineno);
            return;
        }
        entry = entry->next;
    }
    printf("Error: Method %s not defined (line %d)\n", name, yylineno);
    error_flag = 1;
    exit(EXIT_FAILURE);
}
```

Εικόνα 2.2.9

7) Συνάρτηση setMethodName

Η συνάρτηση setMethodName (εικόνα 2.2.10) αντιγράφει το όνομα της μεθόδου σε μία μεταβλητή methodName χρησιμοποιώντας την strdup.

```c
void setMethodName(char* name) {
    methodName = strdup(name);
}
```

Εικόνα 2.2.10

8) Συνάρτηση setVarType

Η συνάρτηση setVarType (εικόνα 2.2.11) αντιγράφει τον τύπο της μεταβλητής var_type χρησιμοποιώντας τη strdup.

```
void setVarType(char* type) {
    var_type = strdup(type);
}
```

Εικόνα 2.2.11

9) Συνάρτηση setMethodType

Η συνάρτηση setMethodType (εικόνα 2.2.12) αντιγράφει τον τύπο της μεθόδου σε μια μεταβλητή method_type χρησιμοποιώντας τη strdup.

```
void setMethodType(char* type) {
    method_type = strdup(type);
}
```

Εικόνα 2.2.12

10) Συνάρτηση setIdentifierName

Η συνάρτηση setIdentifierName (εικόνα 2.2.13) αντιγράφει το όνομα του αναγνωριστικού σε μια μεταβλητή identifier_name χρησιμοποιώντας τη strdup.

```
void setIdentifierName(char* name) {
    identifier_name = strdup(name);
}
```

Εικόνα 2.2.13

11) Συνάρτηση store_variable_value

Η συνάρτηση store_variable_value (εικόνα 2.2.14) αναθέτει μία τιμή σε μία μεταβλητή, είτε ενημερώνοντας μια υπάρχουσα μεταβλητή, είτε δημιουργώντας μια νέα αν δεν βρεθεί.

```c
void store_variable_value(const char* name, int value) {
    SymbolTableEntry* entry = variableTable;
    while (entry != NULL) {
        if (strcmp(entry->name, name) == 0) {
            // Print message indicating the assignment
            printf("Variable '%s' assigned value: %d\n", name, value);
            entry->value = value;
            return;
        }
        entry = entry->next;
    }

    // If the variable is not found, create a new entry
    SymbolTableEntry* newEntry = (SymbolTableEntry*)malloc(sizeof(SymbolTableEntry));
    newEntry->name = strdup(name);
    newEntry->value = value;
    newEntry->next = variableTable;
    variableTable = newEntry;

    // Print message for the new variable assignment
    printf("New variable '%s' assigned value: %d\n", identifier_name, value);
}
```

Εικόνα 2.2.14

12) Συνάρτηση store_variable_boolean_value

Η συνάρτηση store_variable_boolean_value (εικόνα 2.2.15) αναθέτει μία τιμή σε μία μεταβλητή, είτε ενημερώνοντας μια υπάρχουσα μεταβλητή, είτε δημιουργώντας μια νέα αν δεν βρεθεί, όπως ακριβώς και η store_variable_value, απλά για boolean μεταβλητες.

```c
void store_variable_boolean_value(const char* name, int value) {
    SymbolTableEntry* entry = variableTable;
    while (entry != NULL) {
        if (strcmp(entry->name, name) == 0) {
            // Print message indicating the assignment
            printf("Variable '%s' assigned value: %s\n", name, value ? "true" : "false");
            entry->value_boolean = value;
            return;
        }
        entry = entry->next;
    }

    // If the variable is not found, create a new entry
    SymbolTableEntry* newEntry = (SymbolTableEntry*)malloc(sizeof(SymbolTableEntry));
    newEntry->name = strdup(name);
    newEntry->value = 0; // Assuming value is not used for booleans
    newEntry->value_double = 0.0; // Assuming value_double is not used for booleans
    newEntry->value_string = NULL; // Assuming value_string is not used for booleans
    newEntry->value_boolean = value;
    newEntry->next = variableTable;
    variableTable = newEntry;
```

Εικόνα 2.2.15

13) Συνάρτηση store_variable_char_value

Η συνάρτηση store_variable_char_value (εικόνα 2.2.16) αναθέτει μια τιμή τύπου χαρακτήρα σε μία μεταβλητή, είτε ενημερώνοντας μια υπάρχουσα μεταβλητή, είτε δημιουργώντας μια νέα, αν δεν βρεθεί.

```
void store_variable_char_value(const char* name, char value) {
    SymbolTableEntry* entry = variableTable;
    while (entry != NULL) {
        if (strcmp(entry->name, name) == 0) {
            // Print message indicating the assignment
            printf("Variable '%s' assigned value: %c\n", name, value);
            entry->value_char = value;
            return;
        }
        entry = entry->next;
    }

    // If the variable is not found, create a new entry
    SymbolTableEntry* newEntry = (SymbolTableEntry*)malloc(sizeof(SymbolTableEntry));
    newEntry->name = strdup(name);
    newEntry->value = 0; // Assuming value is not used for chars
    newEntry->value_double = 0.0; // Assuming value_double is not used for chars
    newEntry->value_string = NULL; // Assuming value_string is not used for chars
    newEntry->value_char = value;
    newEntry->next = variableTable;
    variableTable = newEntry;

    // Print message for the new variable assignment
    printf("New variable '%s' assigned value: %c\n", name, value);
}
```

Εικόνα 2.2.16

Error Handling

Η συνάρτηση yyerror καλείται όταν εντοπίζει ένα συντακτικό σφάλμα κατά την ανάλυση ενός προγράμματος. Στη αρχή δέχεται μια συμβολοσειρά s η οποία περιέχει το μήνυμα του σφάλματος και εκτυπώνει το μήνυμα που αναφέρει την γραμμή (yylineno) και το μήνυμα του σφάλματος s.
Με την current_token_text εκτυπώνεται το σύμβολο που προκάλεσε σφάλμα, ενώ με το error_flag ορίζουμε να είναι ίσο με 1 για να ενεργοποιηθεί όταν εντοπιστεί σφάλμα.
Με τον κωδικό EXIT_FAILURE τερματίζει το πρόγραμμα. Τα παραπάνω φαίνονται στην εικόνα 2.2.17.

```c
void yyerror(const char *s) {
    printf("Syntax error at line %d: %s\n", yylineno, s);
    printf("Token encountered: %s\n", current_token_text);

    error_flag = 1;
    exit(EXIT_FAILURE); // Exit the program
}
```

Εικόνα 2.2.17

MAIN

Ο σκοπός της συνάρτησης `main` είναι να λειτουργήσει ως το σημείο εισόδου του προγράμματος και να εκτελέσει την ανάλυση ενός αρχείου πηγαίου κώδικα. Αρχικά, ελέγχουμε αν ο χρήστης έχει δώσει τον σωστό αριθμό ορισμάτων. Αν όχι, τυπώνουμε το μήνυμα χρήσης και τερματίζουμε το πρόγραμμα. Στη συνέχεια, προσπαθούμε να ανοίξουμε το αρχείο που έχει δοθεί ως όρισμα και, αν αποτύχει αυτή η προσπάθεια, τυπώνουμε μήνυμα σφάλματος και τερματίζουμε το πρόγραμμα. Αν το άνοιγμα του αρχείου είναι επιτυχές, θέτουμε την `yyin` ώστε να δείχνει στο αρχείο που άνοιξε, ώστε ο Flex να διαβάζει από αυτό το αρχείο. Μετά από αυτή τη ρύθμιση, εκκινούμε την ανάλυση συντακτικού χρησιμοποιώντας τη συνάρτηση `yyparse` που έχει παραχθεί από τον Bison. Αφού ολοκληρωθεί η ανάλυση, κλείνουμε το αρχείο. Τέλος, ελέγχουμε τη σημαία `error_flag` για να αποφασίσουμε αν η μετάφραση ολοκληρώθηκε με επιτυχία ή απέτυχε λόγω συντακτικών σφαλμάτων, τυπώνουμε το αντίστοιχο μήνυμα και επιστρέφουμε τον κατάλληλο κωδικό εξόδου. Τα παραπάνω φαίνονται στην εικόνα 2.2.18.

```c
int main(int argc, char *argv[]) {
    if (argc != 2) {
        printf("Usage: %s file_name\n", argv[0]);
        return 1;
    }

    FILE *fp = fopen(argv[1], "r");
    if (!fp) {
        printf("Error opening file %s\n", argv[1]);
        return 1;
    }

    yyin = fp;

    yyparse();

    fclose(fp);

    if (error_flag) {
        printf("Compilation failed with errors.\n");
        return 1;
    } else {
        printf("Compilation complete with no errors.\n");
        return 0;
    }
    return 0;
}
```

Εικόνα 2.2.18

2.3 Tests

2a) Test code

```java
public class Main {
    public void main(String args) {
        // Assign values to variables
        int x = 20;

        int y = 4;
        int z = 2;
        int x = 5;
        double a = 13.4d;
        boolean b = true;
```

```
        char value = 'a';
        String test = "Hello";
    }
}
```

```
theok_iykl84u@LaptopKatsa /cygdrive/c/Users/theok_iykl84u/OneDrive/Desktop/FLEXB
ISON
$ ./parser test2a.java
Variable 'x' assigned value: 20
Variable x found (line 4)
Variable 'y' assigned value: 4
Variable y found (line 5)
Variable 'z' assigned value: 2
Variable z found (line 6)
Variable 'x' assigned value: 5
Variable x found (line 7)
New variable 'a' assigned value: 13.400000
Variable a found (line 8)
New variable 'b' assigned value: true
Variable b found (line 9)
New variable 'value' assigned value: a
Variable value found (line 10)
New variable 'test' assigned value: "Hello"
Variable test found (line 11)
Method test found (line 12)
Compilation complete with no errors.
```

2b) Test code

```
int x = 15, y = 16, z = 17;

double a = 14.3d, b = 45.2d , c = 53.6d;

String d = "Hello", e = "It", f = "Works";
```

```
theok_iykl84u@LaptopKatsa /cygdrive/c/Users/theok_iykl84u/OneDrive/Desktop/FLEXB
ISON
$ ./parser test2b.java
Variable 'x' assigned value: 15
Variable 'y' assigned value: 16
Variable 'z' assigned value: 17
Variable z found (line 1)
New variable 'a' assigned value: 14.300000
New variable 'b' assigned value: 45.200000
New variable 'c' assigned value: 53.600000
Variable c found (line 2)
New variable 'd' assigned value: "Hello"
New variable 'e' assigned value: "It"
New variable 'f' assigned value: "Works"
Variable f found (line 3)
Compilation complete with no errors.
```

3a) Test code

```
int a = 5;
int b = 10;
private int standaloneMethod(int a, int b) {
    return a;
}

undefinedMethod(); // error undeclared method
class Main {
    public void main(String args) {
        int c = a + x; // error undeclared x
        out.print("The sum is: sum(a, b)");
    }

    public int sum(int x, int y) {
        return x + y;
```

```
    }
}
```

```
theok_iykl84u@LaptopKatsa /cygdrive/c/Users/theok_iykl84u/OneDrive/Desktop/FLEXB
ISON
$ ./parser test3a.java
Variable 'a' assigned value: 5
Variable a found (line 1)
Variable 'b' assigned value: 10
Variable b found (line 2)
Variable a found (line 4)
Method standaloneMethod found (line 5)
Error: Method undefinedMethod not defined (line 7)
```

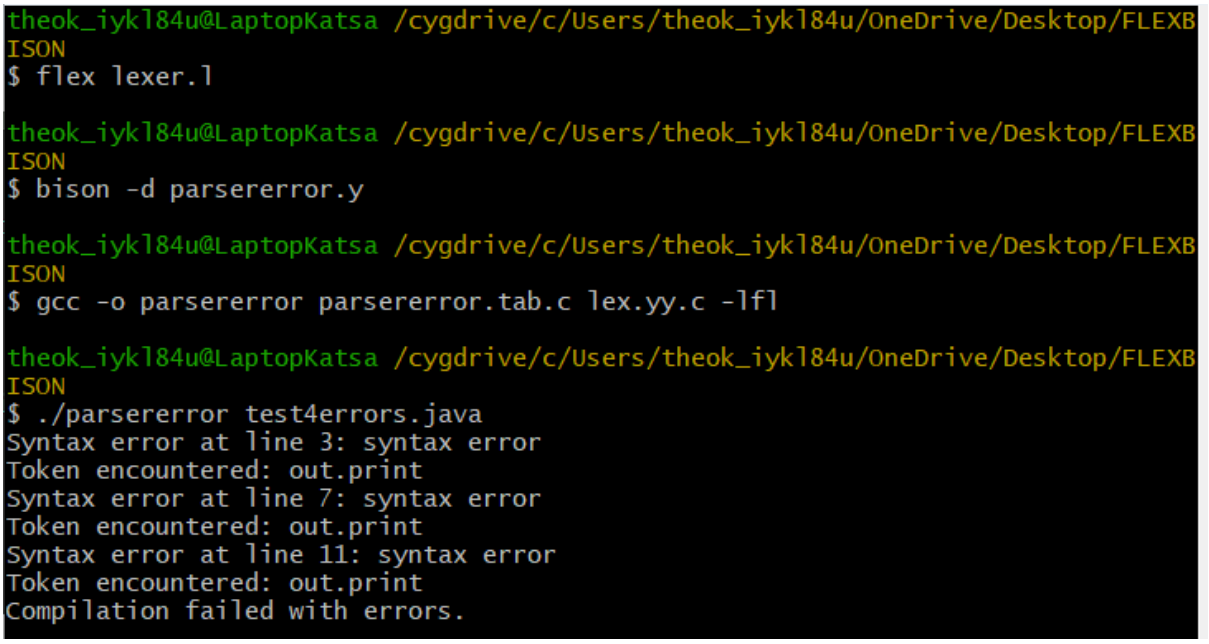
4) Test code

```
class MyClass {
    int myMethod() {
       System.out.println("Hello, world!"); // Error: system
    }

    int mySecondMethod() {
        System.out.println("Hello, world!"); // Error: system
    }

    void main() {
        System.out.println("Hello, world!"); // Error: system
    }
}
```

Parsing with stop on first error

```
theok_iykl84u@LaptopKatsa /cygdrive/c/Users/theok_iykl84u/OneDrive/Desktop/FLEXB
ISON
$ ./parser test4errors.java
Syntax error at line 3: syntax error
Token encountered: out.print

theok_iykl84u@LaptopKatsa /cygdrive/c/Users/theok_iykl84u/OneDrive/Desktop/FLEXB
```

Parsing with all errors appearing



changes made to make parsing continue

```
program:
      | program include_directive
      | program class
      | program method_declaration
      | program variable_declaration
      | program assignment_statement
      | error
      ;
```

error line added

```
statement:
   variable_declaration
   | RETURN SEMICOLON
   | RETURN expression SEMICOLON
   | method_call SEMICOLON { checkMethodCall(methodName); }
   | variable_declaration SEMICOLON
   | assignment_statement
   | object_creation
   | method_declaration
   | print_statement
   | break_statement
   | do_while_statement
   | for_statement
   | switch_statement
   | if_statement
   | error
   ;
```

```
error line added here too

void yyerror(const char *s) {
    printf("Syntax error at line %d: %s\n", yylineno, s);
    printf("Token encountered: %s\n", current_token_text);

    error_flag = 1;
}

    exit(EXIT_FAILURE); // Exit the program–this line was removed
```

2.4 Σχόλια

> Προδιαγραφές
Όλες οι παρακάτω προδιαγραφές της γλώσσας έχουν υλοποιηθεί ορθά, ενώ όπου είναι απαραίτητο έχει παρατεθεί η αντίστοιχη γραμματική.

1) **<u>Class within class και πρώτο γράμμα κεφαλαίο</u>**

```
class:
    opt_modifier CLASS identifier_cap OPEN_BRACE class_body CLOSE_BRACE
    ;

class_body:
    | class_body class_member
    | class_body class
    ;

class_member:
    variable_declaration
    | method_declaration
    | SEMICOLON
    ;

identifier_cap:
    IDENTIFIER {
        if (!isupper(yytext[0])) {
            printf("Error: Class name '%s' must start with a capital letter.\n", yytext);
            error_flag = 1;
        } else {
            setIdentifierName(yytext); // Set the identifier name
        }
```

```
}
;
```

2) Identifiers

Έχει γίνει ανάλυση στην FLEX.

3) Reserved keywords

Μέσω της int_reserved.

4) Data types και String

```
type:
   INT_TYPE { setVarType("int"); }
   | CHAR_TYPE { setVarType("char"); }
   | DOUBLE_TYPE { setVarType("double"); }
   | STRING_TYPE { setVarType("String"); }
   | VOID_TYPE { setVarType("void"); }
   | BOOLEAN_TYPE { setVarType("boolean");}
   ;
```

Και η αντίστοιχη δήλωση τους στην FLEX

5) Δήλωση μεταβλητών

```
variable_declaration:
   opt_modifier type IDENTIFIER {
      setIdentifierName($3);
   } opt_assignment SEMICOLON {
      // Add the variable to the symbol table
      addVariable(identifier_name, assigned_value);
      checkVariableUsage(identifier_name);
   }
   ;
```

6) Δημιουργία ενός αντικειμένου μιας class, πρόσβαση σε μέλη της class

Όπως φαίνεται στο 1.

7) Ορισμός των μεθόδων

```
method_declaration:
   opt_modifier type IDENTIFIER { setIdentifierName($3); setMethodType(var_type);
} OPEN_PAREN argument_list CLOSE_PAREN  OPEN_BRACE statements
CLOSE_BRACE {
      addMethod(identifier_name, method_type);
      checkMethodCall(identifier_name);
```

```
    }
    ;
```

8) Εντολές ανάθεσης (με πράξεις και χωρίς)

```
opt_assignment:
   | ASSIGN INTEGER_LITERAL{
      assigned_value = atoi($2);
      // Print the assigned value
      if (assigned_value != -1) {
         printf("Variable '%s' assigned value: %d\n", identifier_name, assigned_value);
         assigned_value = -1; // Reset assigned value for next variable
      } else {
         // If no assignment, print a message indicating it
         printf("Variable '%s' declared without initialization\n", identifier_name);
      }
      } opt_assignment
   | ASSIGN DOUBLE_LITERAL {
      double assigned_double;
      sscanf($2, "%lf", &assigned_double); // Extract numeric part and convert to
double
      store_variable_double_value($1, assigned_double); // Store the double value
      } opt_assignment
   | ASSIGN STRING_LITERAL {
      assigned_string = strdup($2);
      store_variable_string_value($1, assigned_string);
      }  opt_assignment

   | ASSIGN BOOLEAN_LITERAL  {
      if (strcmp($2, "true") == 0) {
         assigned_value = 0;
      } else {
         assigned_value = 1;
      }
      store_variable_boolean_value($1, assigned_value);
   }  opt_assignment
   | ASSIGN CHARACTER_LITERAL {
      assigned_char = $2[1];
      store_variable_char_value($1, assigned_char);

   } opt_assignment
   | COMMA opt_identifier opt_assignment
   | ASSIGN expression
   ;
```

9) Do while/For

```
do_while_statement:
    DO OPEN_BRACE statements CLOSE_BRACE WHILE OPEN_PAREN
logical_expression CLOSE_PAREN SEMICOLON
    ;

for_statement:
    FOR OPEN_PAREN for_init SEMICOLON for_condition SEMICOLON for_update
CLOSE_PAREN OPEN_BRACE statements CLOSE_BRACE
    ;

for_init:
    variable_declaration
    | assignment_statement
    ;

for_condition:
    logical_expression
    ;

for_update:
    IDENTIFIER ASSIGN expression
    ;
```

10) if else if

```
if_statement:
    IF OPEN_PAREN ifexpression CLOSE_PAREN OPEN_BRACE statements
CLOSE_BRACE optional_else_if optional_else
    ;

optional_else_if:
    | optional_else_if ELSE IF OPEN_PAREN logical_expression CLOSE_PAREN
OPEN_BRACE statements CLOSE_BRACE
    ;

optional_else:
    | ELSE OPEN_BRACE statements CLOSE_BRACE
    ;
```

11) switch cases

```
switch_statement:
    SWITCH OPEN_PAREN expression CLOSE_PAREN OPEN_BRACE
case_clauses default_clause_opt CLOSE_BRACE
```

```
    ;

case_clauses:
    case_clause
    | case_clauses case_clause
    ;

case_clause:
    CASE expression COLON statements
    ;

default_clause_opt:
    | DEFAULT COLON statements
    ;
```

12) out.print

```
print_statement:
    PRINT OPEN_PAREN STRING_LITERAL COMMA expression CLOSE_PAREN
SEMICOLON
    | PRINT OPEN_PAREN STRING_LITERAL CLOSE_PAREN SEMICOLON
    | PRINT OPEN_PAREN IDENTIFIER {setIdentifierName($3);}  CLOSE_PAREN
SEMICOLON {checkVariableUsage(identifier_name);}
    ;
```

13) return

```
return_statement:
    RETURN expression SEMICOLON
    ;
```

14) break

```
break_statement:
    BREAK SEMICOLON
    ;
```

15) single and multi line comments

Φαίνονται στη FLEX.

16) τα κενά και οι χαρακτήρες αλλαγής δεν παίζουν ρόλο

Επίσης έχουν υλοποιηθεί στη FLEX.

> Τι δεν έχει υλοποιηθεί

Η μόνη προδιαγραφή που από άποψη γραμματικής έχει υλοποιηθεί αλλά από άποψη κώδικα δεν λειτουργεί είναι το public private των μεταβλητών/μεθόδων.

```
opt_modifier:
    | PUBLIC
    | PRIVATE
    ;
```

> Όσον αφορά τα ερωτήματα 2α,2β,3α,3β,3γ και 4, λειτουργούν επιτυχώς όλο το 2, το 3α και το 4.
Αναλυτικότερα:

Ερωτήματα 2α,2b

Έχει υλοποιηθεί μέσω της variable_declaration και της opt_assignment οι οποίες παρατίθενται στις προδιαγραφές και λειτουργούν επιτυχώς για όλα τα είδη μεταβλητών (Test2a.java και Test2b.java στην παράγραφο Tests)

Ερώτημα 3α

Επίσης λειτουργεί επιτυχώς καθώς υπάρχουν στον κώδικα έλεγχοι για την δήλωση μιας μεταβλητής και μιας μεθόδου (variable_declaration, method_declaration), όπως επίσης υπάρχει έλεγχος ώστε αν κατά την δήλωση δοθεί όνομα το οποίο ήδη υπάρχει να εμφανίζεται error.
Επιπλέον, στην κλήση μεθόδων υπάρχει έλεγχος για το αν έχει δηλωθεί η μέθοδος που καλείται, αν όχι εμφανίζεται αντίστοιχο μήνυμα.
(Test3a.java στην παράγραφο Tests)

Ερώτημα 3β

Το συγκεκριμένο ερώτημα δεν έχει υλοποιηθεί πλήρως. Έχουν γίνει κάποιες προσπάθειες υλοποίησης του στην γραμματική μας με τη δημιουργία μιας δομής η οποία χρησιμοποιείται γενικά στον κώδικα μας, η SymbolTableEntry:

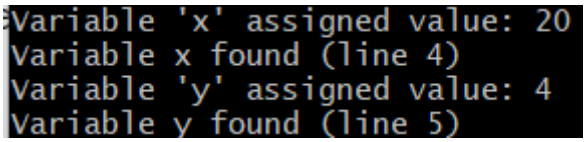
```
typedef struct Scope {
    SymbolTableEntry* symbols;
    struct Scope* next;
} Scope;

Scope* scopeStack = NULL;

void pushScope() {
    Scope* newScope = (Scope*)malloc(sizeof(Scope));
    newScope->symbols = NULL;
    newScope->next = scopeStack;
    scopeStack = newScope;
}
```

```
void popScope() {
   if (scopeStack) {
      Scope* topScope = scopeStack;
      scopeStack = scopeStack->next;
      free(topScope);
   }
}
```

Ωστόσο, δεν μπορούσε το test πρόγραμμα να εμφανίσει τα επιθυμητά error όταν δηλώναμε τις private μεταβλητές εντός μίας κλάσης και προσπαθούσαμε να τις χρησιμοποιήσουμε εκτός αυτής.

Ερώτημα 3γ

Έχουμε χρησιμοποιήσει προειδοποιητικά μηνύματα ώστε κάθε φορά που δηλώνεται μια μεταβλητή να είναι φανερό ότι όντως συμβαίνει και επειτά να εμφανίζεται ένα μήνυμα που μας δείχνει ποια τιμή της έχει ανατεθεί.



Επίσης εάν σε μια παράσταση/ανάθεση υπάρχει μεταβλητή που δεν έχει δηλωθεί ή δεν έχει τιμή εμφανίζεται αντίστοιχο μήνυμα.
Ωστόσο, δεν μπορέσαμε να υλοποιήσουμε την τήρηση των κανόνων προτεραιότητας σε πράξεις.

Ερώτημα 4

Για το ερώτημα αυτό χρησιμοποιούμε τον parsererror και όχι τον parser που έχει υλοποιηθεί για τα υπόλοιπα ερωτήματα.
Οι διαφορές τους είναι απειροελάχιστες απλά όπως φαίνεται στο Test4errors.java στην παράγραφο Tests,αν χρησιμοποιήσουμε τον απλό parser σταματάει στο πρώτο error ενώ με τον parsererror βλέπουμε όλα τα errors, όπως ζητήθηκε.