

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ & ΠΛΗΡΟΦΟΡΙΚΗΣ
ΤΟΜΕΑΣ ΛΟΓΙΚΟΥ ΤΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΜΑΘΗΜΑ: ΑΡΧΕΣ ΓΛΩΣΣΩΝ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ ΚΑΙ ΜΕΤΑΦΡΑΣΤΩΝ
(CEID_NY132)
ΕΑΡΙΝΟ ΕΞΑΜΗΝΟ 2020-2021

**PRINCIPLES OF PROGRAMMING LANGUAGES
AND COMPILER DESIGN 2021:
FLEX & BISON PROJECT
ΤΕΚΜΗΡΙΩΣΗ (DOCUMENTATION) v1.0**

ΑΤΟΜΙΚΗ ΕΠΙΜΕΛΕΙΑ ΕΡΓΑΣΙΑΣ:
ΧΡΗΣΤΟΣ-ΠΑΝΑΓΙΩΤΗΣ ΜΠΑΛΑΤΣΟΥΡΑΣ, (Α.Μ. : 1054335),
Τρέχον έτος σπουδών: 4^{ov}
[email: balatsouras@ceid.upatras.gr]

Κονιάκος Φωκίδος, Απρίλιος 2021

«Αν μπορείς κοίταξε τον φόβο κατάματα και ο φόβος θα φοβηθεί και θα φύγει.»

N. Καζαντζάκης

Πίνακας περιεχομένων

1. ΣΥΣΤΑΣΗ ΤΗΣ ΟΜΑΔΑΣ:	4
2. ΕΙΣΑΓΩΓΙΚΕΣ ΠΛΗΡΟΦΟΡΙΕΣ ΣΧΕΤΙΚΑ ΜΕ ΤΗΝ ΥΛΟΠΟΙΗΣΗ ΤΗΣ ΕΡΓΑΣΙΑΣ	4
3. ΕΚΘΕΣΗ ΚΑΤΑΣΤΑΣΗΣ ΥΛΟΠΟΙΗΣΗΣ ΚΑΙ ΛΕΙΤΟΥΡΓΙΑΣ ΤΩΝ ΤΜΗΜΑΤΩΝ ΤΗΣ ΕΡΓΑΣΙΑΣ	4
4. ΥΛΟΠΟΙΗΣΗ ΕΡΓΑΣΙΑΣ	5
4.1 ΕΡΩΤΗΜΑ 1.a – BNF ΣΥΝΤΑΚΤΙΚΟΣ ΟΡΙΣΜΟΣ	5
ΓΕΝΙΚΟΙ ΚΑΝΟΝΕΣ:	5
ΒΑΣΙΚΟΣ ΚΑΝΟΝΑΣ ΣΥΝΤΑΞΗΣ ΠΡΟΓΡΑΜΜΑΤΟΣ ΣΤΗΝ ΨΕΥΔΟΓΛΩΣΣΑ:	6
ΕΝΑΡΞΗ ΠΡΟΓΡΑΜΜΑΤΟΣ:	6
ΟΡΙΣΜΟΣ ΣΥΝΑΡΤΗΣΕΩΝ:	6
ΔΗΛΩΣΗ ΜΕΤΑΒΛΗΤΩΝ:	6
ΚΥΡΙΟ ΜΕΡΟΣ ΠΡΟΓΡΑΜΜΑΤΟΣ (MAIN):	6
ΕΝΤΟΛΕΣ ΠΡΟΓΡΑΜΜΑΤΟΣ:	7
ΠΑΡΑΔΟΧΕΣ ΣΧΕΔΙΑΣΗΣ ΤΗΣ ΓΡΑΜΜΑΤΙΚΗΣ:	9
4.2 ΕΡΩΤΗΜΑ 1.b – ΥΛΟΠΟΙΗΣΗ ΛΕΚΤΙΚΟΥ ΚΑΙ ΣΥΝΤΑΚΤΙΚΟΥ ΑΝΑΛΥΤΗ	10
4.2.1 ΥΛΟΠΟΙΗΣΗ ΤΟΥ ΛΕΚΤΙΚΟΥ ΑΝΑΛΥΤΗ (LEXER):	11
4.2.2 ΥΛΟΠΟΙΗΣΗ ΤΟΥ ΣΥΝΤΑΚΤΙΚΟΥ ΑΝΑΛΥΤΗ (PARSER):	15
4.2.3 ΠΑΡΑΔΕΙΓΜΑ ΛΕΙΤΟΥΡΓΙΑΣ (TEST CASE):	24
4.3 ΕΡΩΤΗΜΑ 2 – ΔΗΛΩΣΗ ΔΟΜΗΣ (STRUCT)	30
4.3.1 BNF ΟΡΙΣΜΟΣ ΤΟΥ ΣΥΝΤΑΚΤΙΚΟΥ ΤΟΥ ΤΜΗΜΑΤΟΣ ΔΗΛΩΣΗΣ ΔΟΜΗΣ ΚΑΙ ΤΥΠΟΥ ΔΕΔΟΜΕΝΩΝ ΧΡΗΣΤΗ	30
4.3.2 ΕΠΙΚΑΙΡΟΠΟΙΗΣΗ ΤΗΣ ΥΛΟΠΟΙΗΣΗΣ ΤΟΥ ΛΕΞΙΚΟΥ ΚΑΙ ΣΥΝΤΑΚΤΙΚΟΥ ΑΝΑΛΥΤΗ	32
4.3.3 ΠΑΡΑΔΕΙΓΜΑ ΛΕΙΤΟΥΡΓΙΑΣ (TEST CASE):	41
4.4 ΕΡΩΤΗΜΑ 3 – ΕΛΕΓΧΟΣ ΣΩΣΤΗΣ ΔΗΛΩΣΗΣ ΜΕΤΑΒΛΗΤΩΝ ΚΑΙ ΣΥΝΑΡΤΗΣΕΩΝ	46
4.4.1 ΕΝΗΜΕΡΩΣΗ ΤΟΥ ΣΥΝΤΑΚΤΙΚΟΥ ΟΡΙΣΜΟΥ ΤΗΣ ΓΛΩΣΣΑΣ	46
4.4.2 ΤΡΟΠΟΠΟΙΗΣΗ ΤΟΥ ΣΥΝΤΑΚΤΙΚΟΥ ΑΝΑΛΥΤΗ (PARSER)	48
4.4.3 ΠΑΡΑΔΕΙΓΜΑ ΛΕΙΤΟΥΡΓΙΑΣ (TEST CASE):	59
4.5 ΕΡΩΤΗΜΑ 4 – ΣΧΟΛΙΑ ΠΟΛΛΑΠΛΩΝ ΓΡΑΜΜΩΝ	64
4.5.1 ΤΡΟΠΟΠΟΙΗΣΗ ΤΟΥ ΛΕΞΙΚΟΥ ΑΝΑΛΥΤΗ (LEXER)	65
4.5.2 ΠΑΡΑΔΕΙΓΜΑ ΛΕΙΤΟΥΡΓΙΑΣ (TEST CASE):	69
5. ΠΑΡΑΡΤΗΜΑ – ΤΕΛΙΚΑ ΑΡΧΕΙΑ ΕΙΣΟΔΟΥ ΓΙΑ ΤΟ FLEX ΚΑΙ ΤΟ BISON	74
5.1 ΚΩΔΙΚΑΣ ΕΙΣΟΔΟΥ ΣΤΟ FLEX (ΑΡΧΕΙΟ ΕΙΣΟΔΟΥ c-like_lexer.l)	74
5.2 ΚΩΔΙΚΑΣ ΕΙΣΟΔΟΥ ΣΤΟ BISON (ΑΡΧΕΙΟ ΕΙΣΟΔΟΥ c-like_parser.y)	77
6. ΠΑΡΑΡΤΗΜΑ – ΤΕΛΙΚΟ ΑΡΧΕΙΟ ΕΙΣΟΔΟΥ ΣΤΟΝ PARSER ΓΙΑ ΕΛΕΓΧΟ ΤΗΣ ΛΕΙΤΟΥΡΓΙΑΣ ΤΟΥ	85
7. ΒΙΒΛΙΟΓΡΑΦΙΑ	87

1. ΣΥΣΤΑΣΗ ΤΗΣ ΟΜΑΔΑΣ:

Η παρούσα εργασία έχει υλοποιηθεί ατομικά. Συνεπώς, τα στοιχεία του μοναδικού μέλους της ομάδας είναι τα ακόλουθα:

Ονοματεπώνυμο: Χρήστος – Παναγιώτης Μπαλατσούρας

Αριθμός Μητρώου: 1054335

Τρέχον έτος σπουδών: 4^ο

email: balatsouras@ceid.upatras.gr

2. ΕΙΣΑΓΩΓΙΚΕΣ ΠΛΗΡΟΦΟΡΙΕΣ ΣΧΕΤΙΚΑ ΜΕ ΤΗΝ ΥΛΟΠΟΙΗΣΗ ΤΗΣ ΕΡΓΑΣΙΑΣ

Compiler Tools: FLEX (LEX), BISON (YACC)

Λειτουργικό Σύστημα Υλοποίησης και Δοκιμής: Ubuntu 20.04 64 bit

Γλώσσα τεκμηρίωσης: Ελληνικά

Γλώσσα των μηνυμάτων του parser: Αγγλικά

Χρονικό Διάστημα Υλοποίησης: Μάρτιος 2021 – Ιούνιος 2021

Η συγκεκριμένη εργασία υλοποιείται στα πλαίσια του μαθήματος «Αρχές γλωσσών προγραμματισμού και μεταφραστών» και αποτελεί την υποχρεωτική εργασία του μαθήματος. Σε αυτή την εργασία θα αναλυθεί μια ψευδογλώσσα που μοιάζει στη γλώσσα προγραμματισμού ANSI C, άρα μια C-like γλώσσα, η οποία περιγράφεται στην εκφώνηση της εργασίας. Κατά τη διάρκεια της υλοποίησης, θα περιγραφεί η παραπάνω γλώσσα σε συντακτική μορφή BNF και θα υλοποιηθεί ένας λεκτικός και ένας συντακτικός αναλυτής με τη χρήση των εργαλείων FLEX και BISON.

3. ΕΚΘΕΣΗ ΚΑΤΑΣΤΑΣΗΣ ΥΛΟΠΟΙΗΣΗΣ ΚΑΙ ΛΕΙΤΟΥΡΓΙΑΣ ΤΩΝ ΤΜΗΜΑΤΩΝ ΤΗΣ ΕΡΓΑΣΙΑΣ

Παρακάτω, παρουσιάζεται ο πίνακας με την κατάσταση της υλοποίησης των τμημάτων της εργασίας.

Α/Α	ΕΡΓΑΣΙΑ	ΥΛΟΠΟΙΗΣΗ	ΛΕΙΤΟΥΡΓΙΑ	ΠΑΡΑΤΗΡΗΣΕΙΣ
1	ΕΡΩΤΗΜΑ 1.a	ΝΑΙ	ΝΑΙ	
2	ΕΡΩΤΗΜΑ 1.b	ΝΑΙ	ΝΑΙ	
3	ΕΡΩΤΗΜΑ 2	ΝΑΙ	ΝΑΙ	
4	ΕΡΩΤΗΜΑ 3	ΝΑΙ	ΝΑΙ	
5	ΕΡΩΤΗΜΑ 4	ΝΑΙ	ΝΑΙ	

Πίνακας 1 – Τμήματα Υλοποίησης της Εργασίας

4. ΥΛΟΠΟΙΗΣΗ ΕΡΓΑΣΙΑΣ

PROJECT'S GITHUB REPOSITORY:

Η συνολική υλοποίηση της εργασίας βρίσκεται στο Github repository:

https://github.com/takis104/Flex-Bison-Project-2021-CEID_NY132

Το αποθετήριο στο Github είναι ιδιωτικό, καθώς η εργασία δεν έχει εξεταστεί ακόμα. Εάν επιθυμείτε πρόσβαση, παρακαλώ επικοινωνήστε μαζί μου στο email που φαίνεται στο εξώφυλλο αυτής της τεκμηρίωσης για να λάβετε πρόσκληση πρόσβασης στο αποθετήριο.

4.1 ΕΡΩΤΗΜΑ 1.a – BNF ΣΥΝΤΑΚΤΙΚΟΣ ΟΡΙΣΜΟΣ

Σε αυτή την ενότητα παρουσιάζεται ο συντακτικός ορισμός της ψευδογλώσσας σε μορφή BNF μαζί με τις απαραίτητες παραδοχές. Πιο συγκεκριμένα, παρουσιάζονται οι γραμματικοί κανόνες για τη σύνταξη ενός προγράμματος στη ζητούμενη ψευδογλώσσα.

ΓΕΝΙΚΟΙ ΚΑΝΟΝΕΣ:

Εδώ παρουσιάζονται οι γραμματικοί κανόνες που χρησιμοποιούνται συχνά στους ορισμούς των γραμματικών κανόνων της γλώσσας.

<LETTER> ::= <LOWER_CASE> | <UPPER_CASE>

<LOWER_CASE> ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z

<UPPER_CASE> ::= A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z

<DIGIT> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<NUMBER> ::= <DIGIT> | <DIGIT><NUMBER> | <NUMBER> "." <NUMBER>

<BASIC_CHARACTER> ::= "_" | <LETTER> | <DIGIT>

<BASIC_STRING> ::= ε | <BASIC_STRING> <BASIC_CHARACTER>

<NAME> ::= <LETTER> <BASIC_STRING> | "_" <BASIC_STRING>

<ARRAY> ::= "[" <NUMBER> "]"

<VARIABLE> ::= <NAME> | <NAME><ARRAY> | <VARIABLE> "," <VARIABLE>

<LITERAL> ::= <NUMBER> | <STRING_LITERAL> | <CHAR_LITERAL>

<COMPARATIVE_OPERATOR> ::= ">" | "<" | "==" | "!="

<CHAR> ::= Οποιοσδήποτε χαρακτήρας ASCII

<CHAR_SEQ> ::= <CHAR> | <CHAR><CHAR_SEQ>

<CHAR_LITERAL> ::= "'" <CHAR> "'"

<STRING_LITERAL> ::= "\"" <CHAR_SEQ> "\""

ΒΑΣΙΚΟΣ ΚΑΝΟΝΑΣ ΣΥΝΤΑΞΗΣ ΠΡΟΓΡΑΜΜΑΤΟΣ ΣΤΗΝ ΨΕΥΔΟΓΛΩΣΣΑ:

**<PROGRAM_STATEMENT> ::= <PROGRAM_DECLARATION> <MAIN_STATEMENT>
"\\n"
| <PROGRAM_DECLARATION> <FUNCTION_STATEMENT> <MAIN_STATEMENT> "\\n"**

ΕΝΑΡΞΗ ΠΡΟΓΡΑΜΜΑΤΟΣ:

Κάθε πρόγραμμα ξεκινάει με τη δεσμευμένη λέξη «PROGRAM»

<PROGRAM_DECLARATION> ::= "PROGRAM" <NAME> "\\n"

ΟΡΙΣΜΟΣ ΣΥΝΑΡΤΗΣΕΩΝ:

Οι γραμματικοί κανόνες για τον προαιρετικό ορισμό των συναρτήσεων.

**<FUNCTION_STATEMENT> ::= <FUNCTION_DECLARATION> <COMMANDS>
<FUNCTION_END> "\\n"**

**| <FUNCTION_DECLARATION> <VARIABLE_DECLARATION> <COMMANDS>
<FUNCTION_END> "\\n"**

**| <FUNCTION_STATEMENT> <FUNCTION_DECLARATION> <COMMANDS>
<FUNCTION_END> "\\n"**

**| <FUNCTION_STATEMENT> <FUNCTION_DECLARATION>
<VARIABLE_DECLARATION> <COMMANDS> <FUNCTION_END> "\\n"**

<FUNCTION_DECLARATION> ::= "FUNCTION" <FUNCTION_NAME>

<FUNCTION_NAME> ::= <NAME> "(" <VARIABLE> ")"

<FUNCTION_END> ::= "RETURN" <VARIABLE> "END_FUNCTION"

| "RETURN" <NUMBER> "END_FUNCTION"

| "RETURN" <CHAR_LITERAL> "END_FUNCTION"

ΔΗΛΩΣΗ ΜΕΤΑΒΛΗΤΩΝ:

Οι γραμματικοί κανόνες για τη δήλωση των μεταβλητών.

<VARIABLE_DECLARATION> ::= "VARS" <DATATYPE> <VARIABLE> ";" "\\n"

| <VARIABLE_DECLARATION> "VARS" <DATATYPE> <VARIABLE> ";" "\\n"

<DATATYPE> ::= "CHAR" | "INTEGER"

ΚΥΡΙΟ ΜΕΡΟΣ ΠΡΟΓΡΑΜΜΑΤΟΣ (MAIN):

Οι γραμματικοί κανόνες για τον ορισμό του κυρίου μέρους του προγράμματος.

<MAIN_STATEMENT> ::= "STARTMAIN" <COMMANDS> "ENDMAIN"

| "STARTMAIN" <VARIABLE_DECLARATION> <COMMANDS>

"ENDMAIN"

ΕΝΤΟΛΕΣ ΠΡΟΓΡΑΜΜΑΤΟΣ:

Οι γραμματικοί κανόνες για την περιγραφή των εντολών του προγράμματος.

Αρχικά παρουσιάζεται ο γραμματικός κανόνας για κάθε εντολή του προγράμματος.

```
<COMMANDS> ::= <COMMAND> "\n"  
                | <COMMANDS> <COMMAND> "\n"  
<COMMAND> ::= <ASSIGNMENT>  
                | <PRINT>  
                | <LOOP_STATEMENT>  
                | <CONTROL_STATEMENT>  
                | <BREAK_COMMAND>  
<LOOP_STATEMENT> ::= <WHILE_STATEMENT>  
                    | <FOR_STATEMENT>  
<CONTROL_STATEMENT> ::= <IF_STATEMENT>  
                        | <SWITCH_STATEMENT>
```

Στη συνέχεια παρουσιάζονται οι γραμματικοί κανόνες για τις επιμέρους εντολές προγράμματος της ψευδογλώσσας.

1. Εντολή ανάθεσης:

```
<ASSIGNMENT> ::= <VARIABLE> "=" <EXPRESSION> ";"  
                | <ASSIGNMENT> <VARIABLE> "=" <EXPRESSION> ";"
```

```
<EXPRESSION> ::= <NUMBER> | <VARIABLE> | <FUNCTION_NAME>  
                | <EXPRESSION> "+" <EXPRESSION>  
                | <EXPRESSION> "-" <EXPRESSION>  
                | <EXPRESSION> "^" <EXPRESSION>  
                | <EXPRESSION> "*" <EXPRESSION>  
                | <EXPRESSION> "/" <EXPRESSION>  
                | "(" <EXPRESSION> ")"
```

2. Εντολές βρόχου:

2.α Δομή Επανάληψης WHILE:

<WHILE_STATEMENT> ::= "WHILE" "(" <CONDITION> ")" "\n" <COMMANDS>
"ENDWHILE"

<CONDITION> ::= <EXPRESSION> <COMPARATIVE_OPERATOR> <EXPRESSION>
| <EXPRESSION> "AND" <EXPRESSION>
| <EXPRESSION> "OR" <EXPRESSION>
| "(" <CONDITION> ")" <COMPARATIVE_OPERATOR> "(" <CONDITION> ")"
| "(" <CONDITION> ")" "AND" "(" <CONDITION> ")"
| "(" <CONDITION> ")" "OR" "(" <CONDITION> ")"

2.β Δομή Επανάληψης FOR:

<FOR_STATEMENT> ::= "FOR" <NAME> ":" <NUMBER> "TO" <NUMBER> "STEP"
<NUMBER> "\n" <COMMANDS> "ENDFOR"

3. Εντολές ελέγχου:

3.α Δομή Ελέγχου IF:

<IF_STATEMENT> ::= "IF" "(" <CONDITION> ")" "THEN" "\n" <COMMANDS> "ENDIF"
| "IF" "(" <CONDITION> ")" "THEN" "\n" <COMMANDS> <ELSE_IF_STATEMENT>
"ENDIF"
| "IF" "(" <CONDITION> ")" "THEN" "\n" <COMMANDS> <ELSE_STATEMENT> "ENDIF"
| "IF" "(" <CONDITION> ")" "THEN" "\n" <COMMANDS> <ELSE_IF_STATEMENT>
<ELSE_STATEMENT> "ENDIF"
<ELSE_IF_STATEMENT> ::= "ELSEIF" "(" <CONDITION> ")" "\n" <COMMANDS>
| <ELSE_IF_STATEMENT> "ELSEIF" "(" <CONDITION> ")" "\n" <COMMANDS>
<ELSE_STATEMENT> ::= "ELSE" "\n" <COMMANDS>

3.β Δομή Ελέγχου SWITCH:

<SWITCH_STATEMENT> ::= "SWITCH" "(" <EXPRESSION> ")" "\n" <CASE>
"ENDSWITCH"
| "SWITCH" "(" <EXPRESSION> ")" "\n" <CASE> <DEFAULT> "ENDSWITCH"
<CASE> ::= "CASE" "(" <EXPRESSION> ")" ":" "\n" <COMMANDS>
| <CASE> "CASE" "(" <EXPRESSION> ")" ":" "\n" <COMMANDS>
<DEFAULT> ::= "DEFAULT" ":" "\n" <COMMANDS>

4. Εντολές εκτύπωσης:

Ο γραμματικός κανόνας για τον ορισμό της εντολής εκτύπωσης.

<PRINT> ::= "PRINT" "(" <STRING_LITERAL> ")" ";"

| "PRINT" "(" <STRING_LITERAL> "," <VARIABLE> ")" ";"

5. Εντολές τερματισμού βρόχου:

Ο γραμματικός κανόνας για τον ορισμό της εντολής προγράμματος που χρησιμεύει στον τερματισμό ενός βρόχου.

<BREAK_COMMAND> ::= "BREAK" ";"

6. Σχόλια μιας γραμμής:

Ο κανόνας για την παραγωγή σχολίων μιας γραμμής. Τα σχόλια μιας γραμμής ορίζονται ως εξής: Οτιδήποτε βρίσκεται έπειτα από το χαρακτήρα % μέχρι το τέλος της γραμμής.

<LINE_COMMENT> ::= "%" <CHAR_SEQ>

ΠΑΡΑΔΟΧΕΣ ΣΧΕΔΙΑΣΗΣ ΤΗΣ ΓΡΑΜΜΑΤΙΚΗΣ:

1. Το όνομα του προγράμματος που ακολουθεί μετά τη δεσμευμένη λέξη «PROGRAM» ακολουθεί τους κανόνες ονοματολογίας των μεταβλητών της γλώσσας C.
2. Όπως και στη γλώσσα C, τα ονόματα των μεταβλητών ξεκινούν με γράμμα ή με κάτω παύλα και στη συνέχεια ακολουθεί γράμμα, αριθμός ή κάτω παύλα.
3. Η γλώσσα υποστηρίζει τα εξής κυριολεκτικά (literals): αριθμός είτε ακέραιος είτε κινητής υποδιαστολής, χαρακτήρας (char literal) μέσα σε ' ' και ακολουθία χαρακτήρων (string literal) μέσα σε " ".
4. Οι αριθμοί κινητής υποδιαστολής που υποστηρίζονται από τη γλώσσα είναι της μορφής: <ΑΡΙΘΜΟΣ>.<ΑΡΙΘΜΟΣ>
5. Στο τέλος της κάθε συνάρτησης, το τμήμα "RETURN <επιστρεφόμενη τιμή>" και η λέξη «END_FUNCTION» βρίσκονται στην ίδια γραμμή.
6. Στο τέλος της κάθε συνάρτησης, η επιστρεφόμενη τιμή μπορεί να είναι είτε μεταβλητή, είτε αριθμός είτε char literal.
7. Στην παραπάνω γραμματική, με τον όρο "\n" εννοείται ο χαρακτήρας αλλαγής γραμμής (newline).
8. Έχουν προστεθεί στη γραμματική ορισμένες επιπλέον αλλαγές γραμμής μεταξύ των διάφορων εντολών προγράμματος, σε σύγκριση με την περιγραφή της γλώσσας στην εκφώνηση, για αισθητικούς λόγους.
9. Στις εντολές υπό συνθήκη μπορεί τα έντελα της λογικής έκφρασης να είναι και αυτά συγκρίσεις. Σε αυτή την περίπτωση, οι συγκρίσεις πρέπει να είναι μέσα σε παρενθέσεις.

Για παράδειγμα:

IF ((i > 0) AND (k < 1))

4.2 ΕΡΩΤΗΜΑ 1.b – ΥΛΟΠΟΙΗΣΗ ΛΕΚΤΙΚΟΥ ΚΑΙ ΣΥΝΤΑΚΤΙΚΟΥ ΑΝΑΛΥΤΗ

ΣΧΕΤΙΚΑ ΑΡΧΕΙΑ:

Εντός του συμπίεσμένου αρχείου της υποβληθείσας εργασίας, θα βρείτε τα παρακάτω αρχεία:

- **c-like_lexer.l**: Το αρχείο εισόδου του Flex, περιέχει την περιγραφή των λεξημάτων που αναγνωρίζονται από τη ψευδογλώσσα.
- **c-like_parser.y**: Το αρχείο εισόδου του Bison, περιέχει τον ορισμό της γραμματικής της ψευδογλώσσας.
- **prog.c**: Ένα πρόγραμμα γραμμένο στην παραπάνω περιγραφόμενη ψευδογλώσσα που ομοιάζει στη γλώσσα C.
- **output.c**: Το αρχείο εξόδου του λεκτικού και συντακτικού αναλυτή που περιέχει αυτούσιες τις γραμμές του κώδικα του αρχείου "prog.c", οι οποίες αναγνωρίστηκαν ορθά από αυτόν τον αναλυτή.
- **diagnostics.txt**: Ένα αρχείο κειμένου που περιέχει διαγνωστικές πληροφορίες, όπως μηνύματα σφάλματος ή επιτυχίας σχετικά με την ανάλυση του κώδικα του prog.c από τον αναλυτή.
- **a.out**: Ο λεκτικός και συντακτικός αναλυτής σε εκτελέσιμη μορφή.

Χρησιμοποιώντας τα προγράμματα Flex και Bison, για τους σκοπούς της εργασίας, υλοποιήθηκε ένας λεκτικός και ένας συντακτικός αναλυτής, ο οποίος αναγνωρίζει ένα πρόγραμμα γραμμένο στην ψευδογλώσσα που ομοιάζει στη C και επιστρέφει το ίδιο το πρόγραμμα στο τερματικό και σε ένα αρχείο «output.c» μαζί με τα απαραίτητα διαγνωστικά μηνύματα που προκύπτουν από την εκτέλεση του αναλυτή. Κατά παραδοχή, τα διαγνωστικά μηνύματα επιστρέφονται και στο τερματικό και σε ένα εξωτερικό αρχείο κειμένου «diagnostics.txt».

Ο υλοποιηθείς λεκτικός και συντακτικός αναλυτής, στο εξής θα αναφέρεται στο υπόλοιπο της τεκμηρίωσης ως «αναλυτής».

Ο αναλυτής έχει υλοποιηθεί σε λειτουργικό σύστημα Ubuntu Linux 20.04, συνεπώς για να δημιουργηθεί η εκτελέσιμη μορφή του, χρειάζεται να γίνει εκτέλεση των παρακάτω εντολών στο τερματικό:

1. `bison -y -d c-like_parser.y`
2. `flex c-like_lexer.l`
3. `gcc -o myparser y.tab.c lex.yy.c -lfl`
4. `./myparser prog.c`

4.2.1 ΥΛΟΠΟΙΗΣΗ ΤΟΥ ΛΕΚΤΙΚΟΥ ΑΝΑΛΥΤΗ (LEXER):

Ο λεκτικός αναλυτής, αποτελεί μέρος του αναλυτή της εργασίας και υλοποιήθηκε με το εργαλείο Flex.

ΚΩΔΙΚΑΣ ΕΙΣΟΔΟΥ ΣΤΟ FLEX (ΑΡΧΕΙΟ ΕΙΣΟΔΟΥ c-like_lexer.l):

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

%option yylineno

underscore      "_"
digit           [0-9]
letter          [a-zA-Z]
chartype        "CHAR"
inttype         "INTEGER"
comparative     ">" | "<" | "==" | "!="
char            "."
charsequence    {char}*
strliteral      \"{charsequence}\"
charliteral     \"{char}\"
linecomment     \"%{charsequence}
datatype        {chartype}|{inttype}
basiccharacter  {underscore}|{letter}|{digit}
number          {digit}+|{digit}+\".\"{digit}+
basicstring     {basiccharacter}*
name            {letter}{basicstring}|{underscore}{basicstring}
array           \"[{number}]\"

%%

[ \t] { yylval.str = strdup(yytext); ECHO; printf(\"%s\",yytext);}
\"\\n\" { yylval.str = strdup(yytext); ECHO; printf(\"%s\",yytext); return
NEWLINE; }
\";\"   { yylval.str = strdup(yytext); ECHO; printf(\"%s\",yytext); return ';' ; }
\":\"   { yylval.str = strdup(yytext); ECHO; printf(\"%s\",yytext); return ':' ; }
\", \"  { yylval.str = strdup(yytext); ECHO; printf(\"%s\",yytext); return ',' ; }
\"(\"    { yylval.str = strdup(yytext); ECHO; printf(\"%s\",yytext); return '(' ; }
\")\"    { yylval.str = strdup(yytext); ECHO; printf(\"%s\",yytext); return ')' ; }
\"+\"    { yylval.str = strdup(yytext); ECHO; printf(\"%s\",yytext); return '+' ; }
\"-\"    { yylval.str = strdup(yytext); ECHO; printf(\"%s\",yytext); return '-' ; }
```

```
"^" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return '^'; }
"*" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return '*'; }
"/" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return '/'; }
"=" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return '='; }
"AND" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return AND; }
"OR" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return OR; }
":=" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
ASSIGN_OPERATOR; }
"PROGRAM" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
PROGRAM; }
"VARS" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return VARS;}
"FUNCTION" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
FUNCTION; }
"END_FUNCTION" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext);
return END_FUNCTION; }
"RETURN" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
RETURN; }
"WHILE" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
WHILE; }
"ENDWHILE" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
ENDWHILE; }
"FOR" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return FOR; }
"TO" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return TO; }
"STEP" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return STEP;}
"ENDFOR" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
ENDFOR; }
"IF" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return IF; }
"THEN" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return THEN;}
"ELSEIF" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
ELSEIF; }
"ELSE" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return ELSE;
}
"ENDIF" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
ENDIF; }
"SWITCH" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
SWITCH; }
"CASE" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return CASE;}
"DEFAULT" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
DEFAULT; }
"ENDSWITCH" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
ENDSWITCH; }
"PRINT" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
PRINT; }
"BREAK" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
BREAK; }
```

```
"STARTMAIN" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
STARTMAIN; }
"ENDMAIN" { yylval.str = strdup(yytext); ECHO; printf("%s\n",yytext); return
ENDMAIN; }
{strliteral} { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
STRLITERAL; }
{charliteral} { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
CHARLITERAL; }
{linecomment} { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); }
{datatype} { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
DATATYPE; }
{name} { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return NAME;}
{number} { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
NUM;}
{array} { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
ARRAY; }
{comparative} { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
COMP_OPERATOR; }
%%
```

Ο λεκτικός αναλυτής, αναγνωρίζει τα παρακάτω σύμβολα (tokens):

TOKEN	ΠΕΡΙΓΡΑΦΗ
NEWLINE	Χαρακτήρας αλλαγής γραμμής "\n"
AND	Ακολουθία χαρακτήρων: "AND"
OR	Ακολουθία χαρακτήρων: "OR"
ASSIGN_OPERATOR	Ακολουθία χαρακτήρων: "=="
PROGRAM	Ακολουθία χαρακτήρων: "PROGRAM"
VARS	Ακολουθία χαρακτήρων: "VARS"
FUNCTION	Ακολουθία χαρακτήρων: "FUNCTION"
END_FUNCTION	Ακολουθία χαρακτήρων: "END_FUNCTION"
RETURN	Ακολουθία χαρακτήρων: "RETURN"
WHILE	Ακολουθία χαρακτήρων: "WHILE"
ENDWHILE	Ακολουθία χαρακτήρων: "ENDWHILE"
FOR	Ακολουθία χαρακτήρων: "FOR"
TO	Ακολουθία χαρακτήρων: "TO"
STEP	Ακολουθία χαρακτήρων: "STEP"
ENDFOR	Ακολουθία χαρακτήρων: "ENDFOR"
IF	Ακολουθία χαρακτήρων: "IF"
THEN	Ακολουθία χαρακτήρων: "THEN"
ELSEIF	Ακολουθία χαρακτήρων: "ELSEIF"
ELSE	Ακολουθία χαρακτήρων: "ELSE"

ENDIF	Ακολουθία χαρακτήρων: "ENDIF"
SWITCH	Ακολουθία χαρακτήρων: "SWITCH"
CASE	Ακολουθία χαρακτήρων: "CASE"
DEFAULT	Ακολουθία χαρακτήρων: "DEFAULT"
ENDSWITCH	Ακολουθία χαρακτήρων: "ENDSWITCH"
PRINT	Ακολουθία χαρακτήρων: "PRINT"
BREAK	Ακολουθία χαρακτήρων: "BREAK"
STARTMAIN	Ακολουθία χαρακτήρων: "STARTMAIN"
ENDMAIN	Ακολουθία χαρακτήρων: "ENDMAIN"
STRLITERAL	Οποιαδήποτε ακολουθία που αποτελείται από το χαρακτήρα « " », έναν ή περισσότερους χαρακτήρες ASCII και το χαρακτήρα « " » στο τέλος.
CHARLITERAL	Οποιαδήποτε ακολουθία που αποτελείται από το χαρακτήρα « ' », έναν χαρακτήρα ASCII και το χαρακτήρα « ' » στο τέλος.
DATATYPE	Είτε η ακολουθία χαρακτήρων "CHAR" είτε η ακολουθία "INTEGER"
NAME	Οποιαδήποτε ακολουθία που αποτελείται από ένα ή περισσότερα γράμματα, αριθμούς ή το χαρακτήρα κάτω παύλας «_». Ωστόσο, η ακολουθία πρέπει υποχρεωτικά να ξεκινάει είτε με γράμμα είτε με κάτω παύλα «_».
NUM	Οποιαδήποτε ακολουθία αποτελείται από ένα ή περισσότερα ψηφία, ή οποιαδήποτε ακολουθία αποτελείται από ένα ή περισσότερα ψηφία έπειτα ακολουθεί ο χαρακτήρας τελείας «.» και μετά ακολουθούν ένα ή περισσότερα ψηφία.
ARRAY	Οποιαδήποτε ακολουθία που αποτελείται από το χαρακτήρα «[», στη συνέχεια ακολουθεί ένας αριθμός όπως περιγράφεται παραπάνω και τέλος ακολουθεί ο χαρακτήρας «]».
COMP_OPERATOR	Ένας από τους χαρακτήρες: ">", "<", "==", "!="

Επίσης, ο λεκτικός αναλυτής αναγνωρίζει τους χαρακτήρες: ";", ":", ",", "(", ")", "+", "-", "^", "*", "/", "=".

Η ψευδογλώσσα υποστηρίζει σχόλια μιας γραμμής, δηλαδή οτιδήποτε βρίσκεται μετά το χαρακτήρα "%" έως το τέλος της γραμμής. Τα συγκεκριμένα σχόλια αναγνωρίζονται από το λεκτικό αναλυτή με την κανονική έκφραση που αναγνωρίζει ακολουθίες που ξεκινούν με «%» και στη συνέχεια ακολουθούν κανέναν, έναν ή πολλούς χαρακτήρες ASCII εκτός του χαρακτήρα αλλαγής γραμμής. Ο λεκτικός αναλυτής, δεν επιστρέφει κάποιο token όταν εντοπίσει κάποιο σχόλιο μιας γραμμής, ώστε να μην αποτελούν μέρος της σύνταξης του προγράμματος.

4.2.2 ΥΛΟΠΟΙΗΣΗ ΤΟΥ ΣΥΝΤΑΚΤΙΚΟΥ ΑΝΑΛΥΤΗ (PARSER):

Ο συντακτικός αναλυτής, αποτελεί μέρος του αναλυτή της εργασίας και υλοποιήθηκε με το εργαλείο Bison.

ΚΩΔΙΚΑΣ ΕΙΣΟΔΟΥ ΣΤΟ BISON (ΑΡΧΕΙΟ ΕΙΣΟΔΟΥ c-like_parser.y):

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

#define YYERROR_VERBOSE 1

void yyerror(char *);
extern FILE *yyin;
extern FILE *yyout;
extern int yylineno;
int line = 0;
FILE *diagnostics;
char *progr;
%}

%union {
    char *str;
}

%token PROGRAM <str>NAME ARRAY NUM COMP_OPERATOR AND OR STRLITERAL CHARLITERAL
%token VARS DATATYPE FUNCTION END_FUNCTION RETURN
%token STARTMAIN ENDMAIN
%token WHILE ENDWHILE
%token FOR ASSIGN_OPERATOR TO STEP ENDFOR
%token IF THEN ELSEIF ELSE ENDIF
%token SWITCH CASE DEFAULT ENDSWITCH
%token PRINT
%token BREAK
%token NEWLINE
%left ','
%left '+' '-'
%left '*' '/'
%right '^'

%start program

%%
```

```
program: program_declaration main_statement newline { printf("Code Parsed
successfully!\n"); fprintf(diagnostics, "Code Parsed successfully!\n"); }
    | program_declaration function main_statement newline { printf("Code
Parsed successfully!\n"); fprintf(diagnostics, "Code Parsed successfully!\n");
}
    ;

program_declaration: PROGRAM NAME newline { progr = $2; }
    ;

main_statement: STARTMAIN commands ENDMAIN /*{ printf("MAIN Statement
found\n"); }*/
    | STARTMAIN variable_declaration commands ENDMAIN /*{
printf("MAIN Statement found\n"); }*/
    ;

variable_declaration: VARS DATATYPE variable ';' newline /*{ printf("Variable
Statement found\n"); }*/
    | variable_declaration VARS DATATYPE variable ';' newline
/*{ printf("Variable Statement found\n"); }*/
    ;

function: function_declaration commands function_end newline /*{
printf("Function Statement found\n"); }*/
    | function_declaration variable_declaration commands function_end
newline /*{ printf("Function Statement found\n"); }*/
    | function function_declaration commands newline function_end newline
/*{ printf("Function Statement found\n"); }*/
    | function function_declaration variable_declaration commands
function_end newline /*{ printf("Function Statement found\n"); }*/
    ;

function_declaration: FUNCTION function_name newline /*{ printf("Function
Declared\n"); }*/
    ;

function_name: NAME '(' variable ')'
    ;

function_end: RETURN NUM END_FUNCTION
    | RETURN CHARLITERAL END_FUNCTION
    | RETURN variable END_FUNCTION
    ;

command: assignment
    | print_statement
```



```
| loop_statement
| break_command
| control_statement
;

commands: command newline
        | commands command newline
;

assignment: variable '=' expression ';' /*{ printf("Assignment statement
found\n"); }*/
        | assignment variable '=' expression ';' /*{ printf("Assignment
statement found\n"); }*/
;

loop_statement: while_statement
              | for_statement
              ;

break_command: BREAK ';' /*{ printf("Break command found\n"); }*/
;

control_statement: if_statement
                 | switch_statement
                 ;

while_statement: WHILE '(' condition ')' newline commands ENDWHILE /*{
printf("While Loop statement found\n"); }*/
;

condition: expression COMP_OPERATOR expression
        | expression AND expression
        | expression OR expression
        | '(' condition ')' COMP_OPERATOR '(' condition ')'
        | '(' condition ')' AND '(' condition ')'
        | '(' condition ')' OR '(' condition ')'
;

for_statement: FOR NAME ASSIGN_OPERATOR NUM TO NUM STEP NUM newline commands
ENDFOR /*{ printf("For Loop statement found\n"); }*/
;

if_statement: IF '(' condition ')' THEN newline commands ENDIF /*{ printf("If
statement found\n"); }*/
        | IF '(' condition ')' THEN newline commands else_if_statement
ENDIF /*{ printf("If statement found\n"); }*/
```

```
| IF '(' condition ')' THEN newline commands else_statement ENDIF
/*{ printf("If statement found\n"); }*/
| IF '(' condition ')' THEN newline commands else_if_statement
else_statement ENDIF /*{ printf("If statement found\n"); }*/
;

else_if_statement: ELSEIF '(' condition ')' newline commands
| else_if_statement ELSEIF '(' condition ')' newline commands
;

else_statement: ELSE newline commands
;

switch_statement: SWITCH '(' expression ')' newline case ENDSWITCH /*{
printf("Switch statement found\n"); }*/
| SWITCH '(' expression ')' newline case default ENDSWITCH /*{
printf("Switch statement found\n"); }*/
;

case: CASE '(' expression ')' ':' newline commands
| case CASE '(' expression ')' ':' newline commands
;

default: DEFAULT ':' newline commands
;

print_statement: PRINT '(' STRLITERAL ')' ';' /*{ printf("Print Statement
found\n"); }*/
| PRINT '(' STRLITERAL ',' variable ')' ';' /*{ printf("Print
Statement found\n"); }*/
;

expression: literal
| variable
| function_name
| expression '+' expression
| expression '-' expression
| expression '^' expression
| expression '*' expression
| expression '/' expression
| '(' expression ')'
;

variable: NAME
| NAME ARRAY
| variable ',' variable
```

```
        ;

literal: NUM
        | STRLITERAL
        | CHARLITERAL
        ;

newline: NEWLINE { line++; }
        ;

%%

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

int main ( int argc, char **argv )
{
    ++argv; --argc;
    if ( argc > 0 )
        yyin = fopen( argv[0], "r" );
    else
        yyin = stdin;
    yyout = fopen ( "output.c", "w" );

    printf("C-like parser, implemented by Christos-Panagiotis Mpalatsouras,
Student ID = 1054335\n");
    printf("Program source code from parser input is following: \n\n");

    diagnostics = fopen("diagnostics.txt", "w");
    fprintf(diagnostics, "**** START of Diagnostic Messages ****\n\n");

    yyparse ();

    printf("Program Name: %s\n", progr);
    fprintf(diagnostics, "Program: %s\n\n", progr);
    printf("Lines of code parsed: %i\n", line);
    fprintf(diagnostics, "Lines of code parsed: %i\n", line);
    fprintf(diagnostics, "\n**** END of Diagnostic Messages ****\n");
    fclose(diagnostics);

    return 0;
}
```

Ο συντακτικός αναλυτής, αναγνωρίζει τους παρακάτω κανόνες της γραμματικής της γλώσσας κατ' αντιστοιχία με την περιγραφή της γραμματικής της γλώσσας σε BNF από το ερώτημα 1.α:

ΚΑΝΟΝΑΣ	ΠΕΡΙΓΡΑΦΗ
program	Αντιστοιχεί στον κανόνα <PROGRAM_STATEMENT> του ερωτήματος 1.α Αποτελεί τον γενικό κανόνα για τη σύνταξη ενός πλήρους προγράμματος.
program_declaration	Αντιστοιχεί στον κανόνα <PROGRAM_DECLARATION> του ερωτήματος 1.α Αποτελεί τον κανόνα για τη δήλωση ενός προγράμματος
main_statement	Αντιστοιχεί στον κανόνα <MAIN_STATEMENT> του ερωτήματος 1.α Αποτελεί τον κανόνα για τη σύνταξη του κυρίου μέρους του προγράμματος.
variable_declaration	Αντιστοιχεί στον κανόνα <VARIABLE_DECLARATION> του ερωτήματος 1.α Αποτελεί τον κανόνα για τη σύνταξη της προαιρετικής δήλωσης μεταβλητών
function	Αντιστοιχεί στον κανόνα <FUNCTION_STATEMENT> του ερωτήματος 1.α Αποτελεί τον κανόνα για τη σύνταξη του ορισμού συναρτήσεων.
function_declaration	Αντιστοιχεί στον κανόνα <FUNCTION_DECLARATION> του ερωτήματος 1.α Αποτελεί τον κανόνα για τη σύνταξη της δήλωσης συναρτήσεων.
function_name	Αντιστοιχεί στον κανόνα <FUNCTION_NAME> του ερωτήματος 1.α Αποτελεί τον κανόνα για τη σύνταξη του τμήματος της δήλωσης μιας συνάρτησης που περιέχει το όνομα και τα ορίσματα μιας συνάρτησης.
function_end	Αντιστοιχεί στον κανόνα <FUNCTION_END> του ερωτήματος 1.α Αποτελεί τον κανόνα για τη σύνταξη του τέλους συναρτήσεων.
command	Αντιστοιχεί στον κανόνα <COMMAND> του ερωτήματος 1.α Αποτελεί τον κανόνα για τη σύνταξη μιας εντολής προγράμματος.

commands	Αντιστοιχεί στον κανόνα <COMMANDS> του ερωτήματος 1.a Αποτελεί τον κανόνα για την ύπαρξη πολλών εντολών προγράμματος μέσα στο πρόγραμμα.
assignment	Αντιστοιχεί στον κανόνα <ASSIGNMENT> του ερωτήματος 1.a Αποτελεί τον κανόνα για τη σύνταξη μιας εντολής ανάθεσης.
loop_statement	Ο κανόνας που περιγράφει εντολές επανάληψης, δηλαδή εντολές while ή for.
break_command	Αντιστοιχεί στον κανόνα <BREAK_COMMAND> του ερωτήματος 1.a Αποτελεί τον κανόνα για τη σύνταξη μιας εντολής τερματισμού βρόχου.
control_statement	Ο κανόνας που περιγράφει εντολές ελέγχου, δηλαδή εντολές if ή switch.
while_statement	Αντιστοιχεί στον κανόνα <WHILE_STATEMENT> του ερωτήματος 1.a Αποτελεί τον κανόνα για τη σύνταξη μιας εντολής επανάληψης while.
condition	Αντιστοιχεί στον κανόνα <CONDITION> του ερωτήματος 1.a Αποτελεί τον κανόνα για τη σύνταξη της έκφρασης της συνθήκης για τις υπό συνθήκη εντολές.
for_statement	Αντιστοιχεί στον κανόνα <FOR_STATEMENT> του ερωτήματος 1.a Αποτελεί τον κανόνα για τη σύνταξη μιας εντολής επανάληψης for.
if_statement	Αντιστοιχεί στον κανόνα <IF_STATEMENT> του ερωτήματος 1.a Αποτελεί τον κανόνα για τη σύνταξη μιας εντολής ελέγχου if.
else_if_statement	Αντιστοιχεί στον κανόνα <ELSE_IF_STATEMENT> του ερωτήματος 1.a Αποτελεί τον κανόνα για τη σύνταξη του τμήματος else if της εντολής if.
else_statement	Αντιστοιχεί στον κανόνα <ELSE_STATEMENT> του ερωτήματος 1.a Αποτελεί τον κανόνα για τη σύνταξη του τμήματος else της εντολής if.
switch_statement	Αντιστοιχεί στον κανόνα <SWITCH_STATEMENT> του ερωτήματος 1.a

	Αποτελεί τον κανόνα για τη σύνταξη μιας εντολής ελέγχου switch.
case	Αντιστοιχεί στον κανόνα <CASE> του ερωτήματος 1.a Αποτελεί τον κανόνα για τη σύνταξη του τμήματος case της εντολής switch.
default	Αντιστοιχεί στον κανόνα <DEFAULT> του ερωτήματος 1.a Αποτελεί τον κανόνα για τη σύνταξη του τμήματος default της εντολής switch.
print_statement	Αντιστοιχεί στον κανόνα <PRINT> του ερωτήματος 1.a Αποτελεί τον κανόνα για τη σύνταξη μιας εντολής εκτύπωσης.
expression	Αντιστοιχεί στον κανόνα <EXPRESSION> του ερωτήματος 1.a Αποτελεί τον κανόνα για τη σύνταξη των εκφράσεων στις εντολές ανάθεσης.
variable	Αντιστοιχεί στον κανόνα <VARIABLE> του ερωτήματος 1.a Αποτελεί τον κανόνα για τη σύνταξη του ονόματος μιας μεταβλητής.
literal	Αντιστοιχεί στον κανόνα <LITERAL> του ερωτήματος 1.a Αποτελεί τον κανόνα για τη σύνταξη των κυριολεκτικών.
newline	Ο κανόνας ενεργοποιείται μόλις εντοπιστεί το token "NEWLINE"

Η συντακτική ανάλυση ξεκινάει με τον κανόνα **program**.

Στο συντακτικό αναλυτή, υπάρχει η οδηγία `#define YYERROR_VERBOSE 1`, η οποία δίνει τη δυνατότητα να εμφανίζονται σφάλματα με περισσότερες λεπτομέρειες όταν η σύνταξη του προγράμματος στην είσοδο δεν είναι σωστή. Επίσης σε αυτά τα σφάλματα εμφανίζεται και ο αριθμός γραμμής στην οποία βρίσκεται το σφάλμα με τη χρήση της μεταβλητής `extern int yylineno` που δηλώνεται στην αρχή του κώδικα.

Με τη δήλωση `%union`, ορίζονται τύποι δεδομένων για τα attributes ορισμένων tokens.

Με τη δήλωση `extern FILE *yyin` ορίζεται ο δείκτης προς το αρχείο εισόδου του αναλυτή και με τη δήλωση `extern FILE *yyout` ορίζεται ο δείκτης προς το αρχείο εξόδου στο οποίο επιστρέφεται το ίδιο το πρόγραμμα από την είσοδο. Το αρχείο εξόδου έχει οριστεί να ονομάζεται «output.c».

Επίσης, με τη δήλωση `FILE *diagnostics`, ορίζεται ο δείκτης προς το αρχείο εξόδου «diagnostics.txt» που περιέχει διαγνωστικές πληροφορίες, όπως μηνύματα σφάλματος ή μηνύματα επιτυχούς αναγνώρισης της εισόδου.

Εντός της συνάρτησης `main`, υπάρχουν οι συναρτήσεις που διαχειρίζονται την είσοδο και τα αρχεία της εξόδου, ενώ επίσης καλείται η `yyparse()` η οποία αναγνωρίζει την είσοδο, παράγει το `parse tree` και εκτελεί τις ενέργειες που περιγράφονται στο πρόγραμμα. Στη συνέχεια, υπάρχουν οι συναρτήσεις που εκτυπώνουν τα διαγνωστικά μηνύματα.

Τα μηνύματα σφάλματος εκτυπώνονται μέσω της συνάρτησης `yyerror()`.

Ακολουθεί στην επόμενη σελίδα το παράδειγμα λειτουργίας του λεξικού και συντακτικού αναλυτή.

4.2.3 ΠΑΡΑΔΕΙΓΜΑ ΛΕΙΤΟΥΡΓΙΑΣ (TEST CASE):

Για τους σκοπούς της δοκιμής του αναλυτή που υλοποιήθηκε στα πλαίσια της εργασίας, συνετάχθη ένα πρόγραμμα γραμμένο στην ανωτέρω ψευδογλώσσα.

Το δοκιμαστικό πρόγραμμα προσεγγίζει την υλοποίηση δυο αλγορίθμων ταξινόμησης, τον bubble sort και τον insertion sort ως συναρτήσεις, οι οποίες καλούνται στο κύριο μέρος του προγράμματος, με τρόπο ώστε να ικανοποιούνται όλοι οι περιορισμοί των προδιαγραφών της ψευδογλώσσας.

Ο κώδικας του παραδείγματος βρίσκεται στο αρχείο prog.c στο φάκελο με τον κώδικα της υλοποίησης του αναλυτή σε flex και bison.

ΚΩΔΙΚΑΣ ΤΟΥ ΠΑΡΑΔΕΙΓΜΑΤΟΣ:

```
PROGRAM Sorting_Algorithms %This is a test programm for lexer and parser
development purposes
FUNCTION BubbleSort(K[20], m)
    VARS INTEGER i,j,head,temp;
    i = 0;
    j = 0;
    head = m;
    WHILE (head > 1)
        j = 0;
        head2 = head - 1;
        PRINT("Sorting elements 0 --> %d, ", head2);
        FOR i:=0 TO 20 STEP 1
            IF (K_i > K_i_plus_1) THEN
                temp = K_i_plus_1;
                K_i_plus_1 = K_i;
                K_i = temp;
                j = j + 1;
            ENDIF
        ENDFOR
        PRINT("Element swaps in the current table scan: %d\n", j);
        IF (j == 0) THEN
            PRINT("No swap during last table scan. The table sorting
completed successfully\n");
            BREAK;
        ENDIF
        head = head - 1;
    ENDWHILE
RETURN K_m END_FUNCTION
FUNCTION InsertionSort(K[20], m)
    VARS INTEGER i,j,k;
    i = 0;
```



```
j = 0;
k = 0;
FOR j:=2 TO 20 STEP 1
    PRINT("Sorting elements 0 --> %d\n", j);
    k = K_j;
    i = j - 1;
    WHILE ((i > 0) AND (k < K_i))
        K_i_plus_1 = K_i;
        i = i - 1;
    ENDWHILE
    K_i_plus_1 = k;
ENDFOR
RETURN K_m END_FUNCTION
STARTMAIN VARS INTEGER n,i,c,A[20];
    n = 20;
    A[20] = values;
    PRINT("Initial Table: \n");
    FOR i:=0 TO 20 STEP 1
        PRINT("%d ", A_i);
    ENDFOR
    PRINT("\nOptions: \n");
    PRINT("1. Bubble-Sort:\n");
    PRINT("2. Insertion-Sort:\n");
    PRINT("3. Selection-Sort:\n");
    PRINT("Select Algorithm >> "); %scanf an option from keyboard
    SWITCH (option)
    CASE (1):
        A_n = BubbleSort(A,n);
        PRINT("Results: \n");
        FOR i:=0 TO 20 STEP 1
            PRINT("%d ", A_i);
        ENDFOR
        PRINT("\n");
    CASE (2):
        A_n = InsertionSort(A,n);
        PRINT("Results: \n");
        FOR i:=0 TO 20 STEP 1
            PRINT("%d ", A_i);
        ENDFOR
        PRINT("\n");
    DEFAULT:
        PRINT("No option selected\n");
    ENDSWITCH
ENDMAIN
```

ΠΑΡΑΔΕΙΓΜΑ ΕΠΙΤΥΧΟΥΣ ΕΚΤΕΛΕΣΗΣ:

Δίνοντας ως είσοδο τον παραπάνω δοκιμαστικό κώδικα στον αναλυτή που υλοποιήθηκε στα πλαίσια της εργασίας, προκύπτει το παρακάτω αποτέλεσμα στα στιγμιότυπα οθόνης που ακολουθούν:

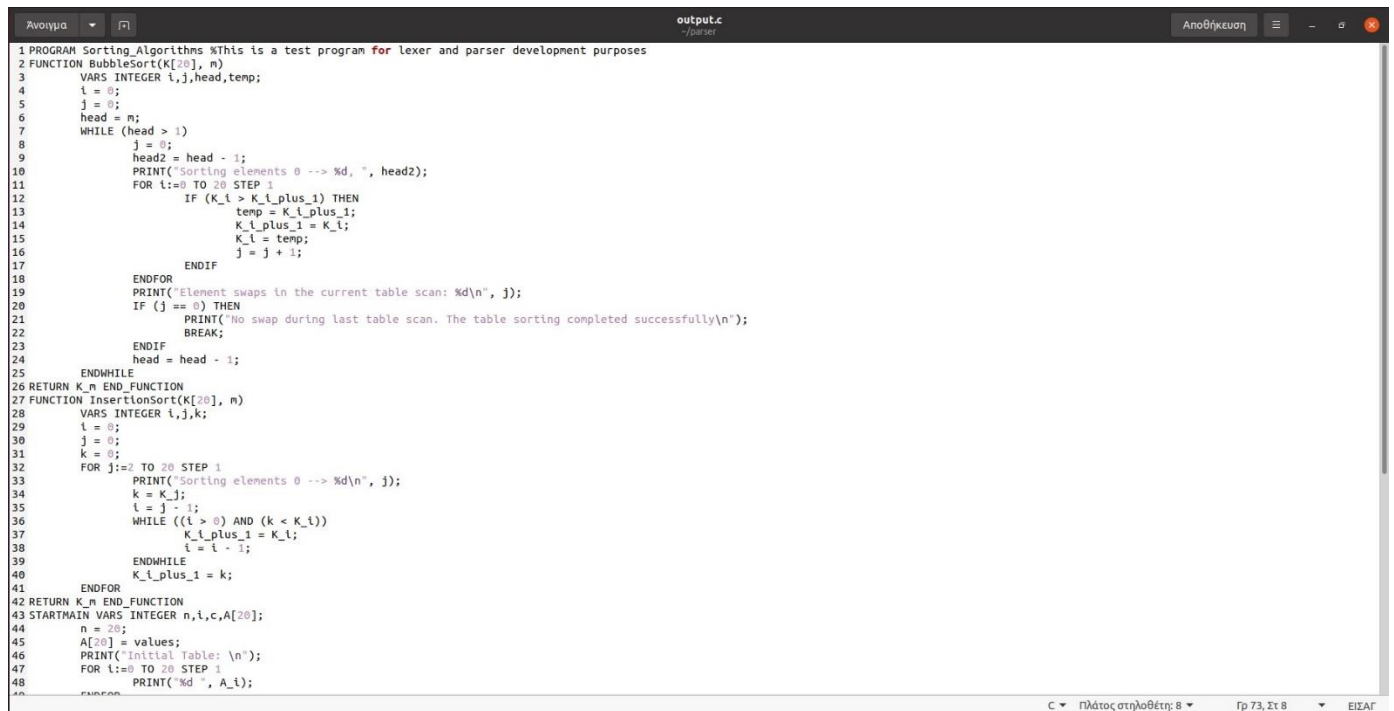
```
takis@takis-VirtualBox: ~/parser
takis@takis-VirtualBox:~/parser$ bison -y -d c-like_parser.y
takis@takis-VirtualBox:~/parser$ flex c-like_lexer.l
takis@takis-VirtualBox:~/parser$ gcc y.tab.c lex.yy.c -lfl
y.tab.c: in function 'yyparse':
y.tab.c:1477:16: warning: implicit declaration of function 'yylex' [-Wimplicit-function-declaration]
1477 |         yychar = yylex();
      |         ^~~~~~
y.tab.c:1667:18: warning: passing argument 1 of 'yyerror' discards 'const' qualifier from pointer target type [-Wdiscarded-qualifiers]
1667 |         yyerror(ymsgp);
      |         ^~~~~~
c-like_parser.y:9:14: note: expected 'char *' but argument is of type 'const char *'
9 | void yyerror(char *);
  | ~~~~~~
takis@takis-VirtualBox:~/parser$ ./a.out prog.c
c-like parser, implemented by Christos-Panagiotis Mpalatsouras, Student ID = 1054335
Program source code from parser input is following:

PROGRAM Sorting Algorithms %This is a test program for lexer and parser development purposes
FUNCTION BubbleSort(K[20], m)
  VARS INTEGER i,j,head,temp;
  i = 0;
  j = 0;
  head = m;
  WHILE (head > 1)
    j = 0;
    head2 = head - 1;
    PRINT("Sorting elements 0 --> %d, ", head2);
    FOR i:=0 TO 20 STEP 1
      IF (K_i > K_i_plus_1) THEN
        temp = K_i_plus_1;
        K_i_plus_1 = K_i;
        K_i = temp;
        j = j + 1;
      ENDIF
    ENDFOR
    PRINT("Element swaps in the current table scan: %d\n", j);
    IF (j == 0) THEN
      PRINT("No swap during last table scan. The table sorting completed successfully\n");
      BREAK;
    ENDIF
    head = head - 1;
  ENDWHILE
RETURN K.m END_FUNCTION
FUNCTION InsertionSort(K[20], m)
  VARS INTEGER i,j,k;
  i = 0;
  j = 0;
  k = 0;
  FOR j:=2 TO 20 STEP 1
    PRINT("Sorting elements 0 --> %d\n", j);
```

```
      j = 0;
      k = 0;
      FOR j:=2 TO 20 STEP 1
        PRINT("Sorting elements 0 --> %d\n", j);
        k = K.j;
        i = j - 1;
        WHILE ((i > 0) AND (k < K_i))
          K_i_plus_1 = K_i;
          i = i - 1;
        ENDWHILE
        K_i_plus_1 = k;
      ENDFOR
RETURN K.m END_FUNCTION
STARTMAIN VARS INTEGER n,i,c,A[20];
n = 20;
A[20] = values;
PRINT("Initial Table: \n");
FOR i:=0 TO 20 STEP 1
  PRINT("%d ", A_i);
ENDFOR
PRINT("\nOptions: \n");
PRINT("1. Bubble-Sort:\n");
PRINT("2. Insertion-Sort:\n");
PRINT("3. Selection-Sort:\n");
PRINT("Select Algorithm >> "); %scanf an option from keyboard
SWITCH (option)
CASE (1):
  A.n = BubbleSort(A,n);
  PRINT("Results: \n");
  FOR i:=0 TO 20 STEP 1
    PRINT("%d ", A_i);
  ENDFOR
  PRINT("\n");
CASE (2):
  A.n = InsertionSort(A,n);
  PRINT("Results: \n");
  FOR i:=0 TO 20 STEP 1
    PRINT("%d ", A_i);
  ENDFOR
  PRINT("\n");
DEFAULT:
  PRINT("No option selected\n");
ENDSWITCH
ENDMAIN
Code Parsed successfully!
Program Name: Sorting_Algorithms
Lines of code parsed: 73
takis@takis-VirtualBox:~/parser$
```

Επειδή το παραπάνω αποτέλεσμα της εκτέλεσης του αναλυτή δεν χώραγε σε μια οθόνη, παραπάνω παρουσιάζονται δύο στιγμιότυπα οθόνης όπου το δεύτερο αποτελεί συνέχεια του πρώτου.

Ο κώδικας της εισόδου, έχει επιστραφεί στην έξοδο στο παραπάνω τερματικό και στο αρχείο output.c:



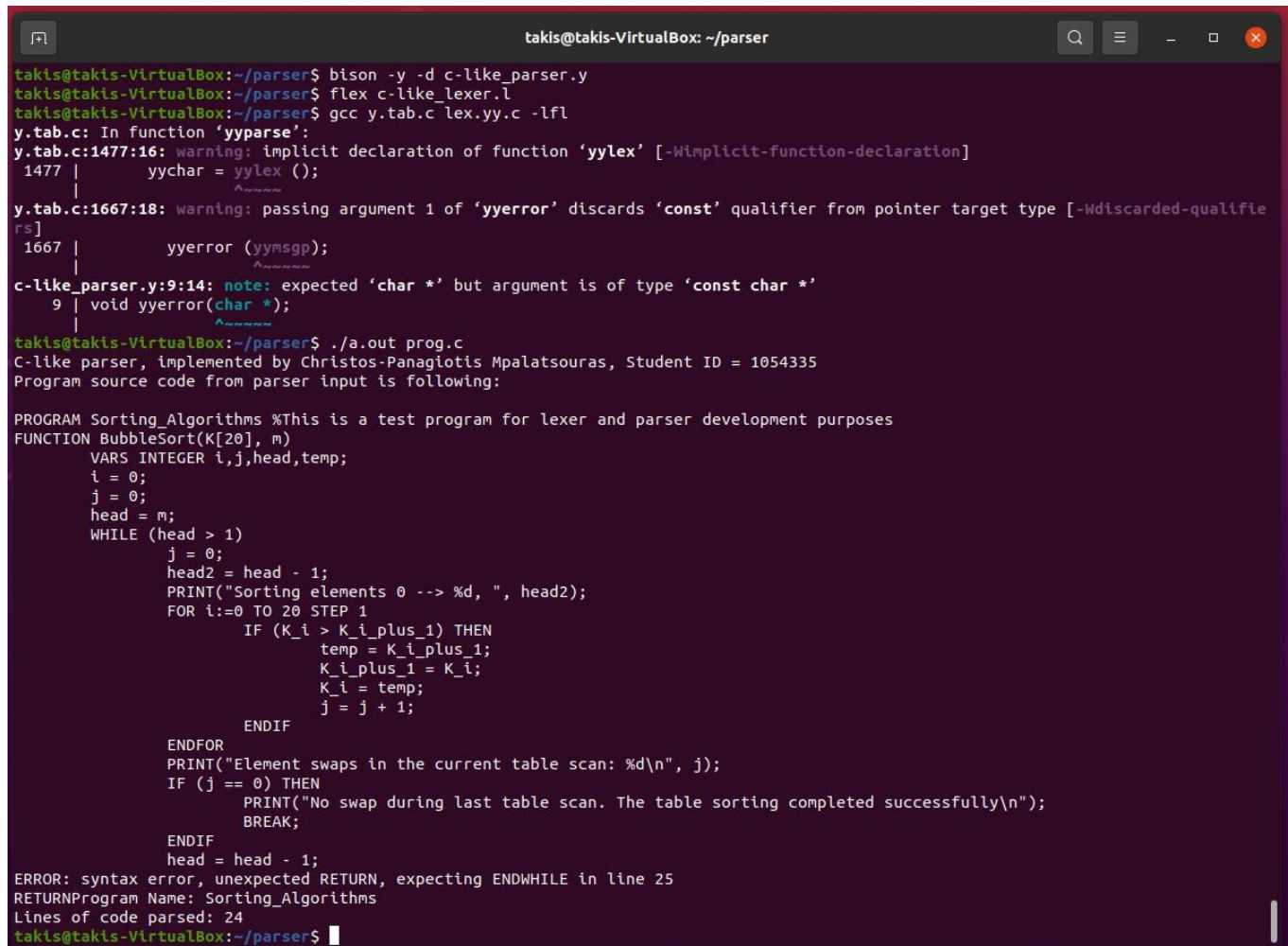
```
1 PROGRAM Sorting_Algorithms %This is a test program for lexer and parser development purposes
2 FUNCTION BubbleSort(K[20], n)
3   VARS INTEGER i,j,head,temp;
4   i = 0;
5   j = 0;
6   head = n;
7   WHILE (head > 1)
8     j = 0;
9     head2 = head - 1;
10    PRINT("Sorting elements 0 --> %d, ", head2);
11    FOR i:=0 TO 20 STEP 1
12      IF (K_i > K_i_plus_1) THEN
13        temp = K_i_plus_1;
14        K_i_plus_1 = K_i;
15        K_i = temp;
16        j = j + 1;
17      ENDIF
18    ENDFOR
19    PRINT("Element swaps in the current table scan: %d\n", j);
20    IF (j == 0) THEN
21      PRINT("No swap during last table scan. The table sorting completed successfully\n");
22      BREAK;
23    ENDIF
24    head = head - 1;
25  ENDWHILE
26 RETURN K,n END_FUNCTION
27 FUNCTION InsertionSort(K[20], n)
28   VARS INTEGER i,j,k;
29   i = 0;
30   j = 0;
31   k = 0;
32   FOR j:=2 TO 20 STEP 1
33     PRINT("Sorting elements 0 --> %d\n", j);
34     k = K_j;
35     i = j - 1;
36     WHILE ((i > 0) AND (k < K_i))
37       K_i_plus_1 = K_i;
38       i = i - 1;
39     ENDWHILE
40     K_i_plus_1 = k;
41   ENDFOR
42 RETURN K,n END_FUNCTION
43 STARTMAIN VARS INTEGER n,i,c,A[20];
44   n = 20;
45   A[20] = values;
46   PRINT("Initial Table: \n");
47   FOR i:=0 TO 20 STEP 1
48     PRINT("%d ", A_i);
49   ENDFOR
```

Ακολουθεί στην επόμενη σελίδα το παράδειγμα ανεπιτυχούς εκτέλεσης.

ΠΑΡΑΔΕΙΓΜΑ ΑΝΕΠΙΤΥΧΟΥΣ ΕΚΤΕΛΕΣΗΣ:

Έστω ότι από τη γραμμή 25 λείπει η λέξη «ENDWHILE».

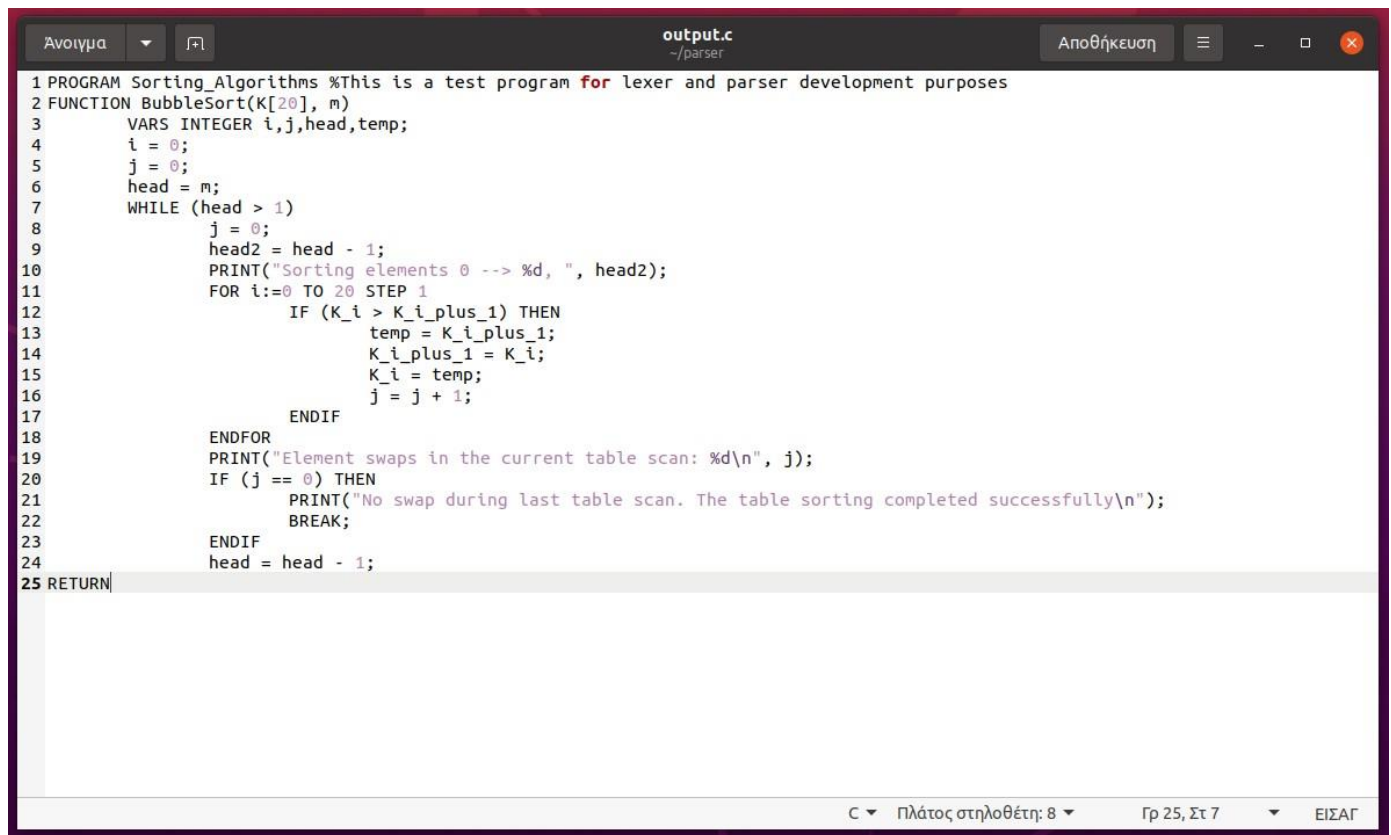
Εμφανίζεται το αντίστοιχο μήνυμα σφάλματος στο τερματικό, όπως φαίνεται στο παρακάτω στιγμιότυπο οθόνης:



```
takis@takis-VirtualBox: ~/parser
takis@takis-VirtualBox:~/parser$ bison -y -d c-like_parser.y
takis@takis-VirtualBox:~/parser$ flex c-like_lexer.l
takis@takis-VirtualBox:~/parser$ gcc y.tab.c lex.yy.c -lfl
y.tab.c: In function 'yyparse':
y.tab.c:1477:16: warning: implicit declaration of function 'yylex' [-Wimplicit-function-declaration]
1477 |         yychar = yylex ();
      |
y.tab.c:1667:18: warning: passing argument 1 of 'yyerror' discards 'const' qualifier from pointer target type [-Wdiscarded-qualifiers]
1667 |         yyerror (yymsgp);
      |
c-like_parser.y:9:14: note: expected 'char *' but argument is of type 'const char *'
  9 | void yyerror(char *);
    |
takis@takis-VirtualBox:~/parser$ ./a.out prog.c
C-like parser, implemented by Christos-Panagiotis Mpalatsouras, Student ID = 1054335
Program source code from parser input is following:

PROGRAM Sorting_Algorithms %This is a test program for lexer and parser development purposes
FUNCTION BubbleSort(K[20], m)
  VARS INTEGER i,j,head,temp;
  i = 0;
  j = 0;
  head = m;
  WHILE (head > 1)
    j = 0;
    head2 = head - 1;
    PRINT("Sorting elements 0 --> %d, ", head2);
    FOR i:=0 TO 20 STEP 1
      IF (K_i > K_i_plus_1) THEN
        temp = K_i_plus_1;
        K_i_plus_1 = K_i;
        K_i = temp;
        j = j + 1;
      ENDIF
    ENDFOR
    PRINT("Element swaps in the current table scan: %d\n", j);
    IF (j == 0) THEN
      PRINT("No swap during last table scan. The table sorting completed successfully\n");
      BREAK;
    ENDIF
    head = head - 1;
  ENDWHILE
RETURN
Program Name: Sorting_Algorithms
Lines of code parsed: 24
takis@takis-VirtualBox:~/parser$
```

Ο κώδικας της εισόδου δεν έχει επιστραφεί ολόκληρος στην έξοδο, στο παραπάνω τερματικό και στο αρχείο output.c, καθώς έχει σταματήσει στη γραμμή που εμφανίζεται το σφάλμα.



```
1 PROGRAM Sorting_Algorithms %This is a test program for lexer and parser development purposes
2 FUNCTION BubbleSort(K[20], m)
3   VARS INTEGER i,j,head,temp;
4   i = 0;
5   j = 0;
6   head = m;
7   WHILE (head > 1)
8     j = 0;
9     head2 = head - 1;
10    PRINT("Sorting elements 0 --> %d, ", head2);
11    FOR i:=0 TO 20 STEP 1
12      IF (K_i > K_i_plus_1) THEN
13        temp = K_i_plus_1;
14        K_i_plus_1 = K_i;
15        K_i = temp;
16        j = j + 1;
17      ENDIF
18    ENDFOR
19    PRINT("Element swaps in the current table scan: %d\n", j);
20    IF (j == 0) THEN
21      PRINT("No swap during last table scan. The table sorting completed successfully\n");
22      BREAK;
23    ENDIF
24    head = head - 1;
25 RETURN
```

4.3 ΕΡΩΤΗΜΑ 2 – ΔΗΛΩΣΗ ΔΟΜΗΣ (STRUCT)

Σε αυτή την ενότητα επεκτείνεται ο λεκτικός και συντακτικός αναλυτής που υλοποιήθηκε στο πρώτο ερώτημα, ώστε να υποστηρίζει τη δήλωση τύπου δεδομένων χρήστη και τη δήλωση δομής. Το συγκεκριμένο τμήμα του προγράμματος, δηλαδή η δήλωση τύπου δεδομένων του χρήστη και η δήλωση δομής, βρίσκεται πριν από τον προαιρετικό ορισμό των συναρτήσεων.

4.3.1 BNF ΟΡΙΣΜΟΣ ΤΟΥ ΣΥΝΤΑΚΤΙΚΟΥ ΤΟΥ ΤΜΗΜΑΤΟΣ ΔΗΛΩΣΗΣ ΔΟΜΗΣ ΚΑΙ ΤΥΠΟΥ ΔΕΔΟΜΕΝΩΝ ΧΡΗΣΤΗ

Ακολουθεί ο συντακτικός ορισμός του τμήματος της δήλωσης δομής και τύπου δεδομένων χρήστη.

```
<STRUCT_STATEMENT> ::= "STRUCT" <NAME> "\n" <STRUCT_VARIABLE>
"ENDSTRUCT"
| "STRUCT" <NAME> "\n" <STRUCT_VARIABLE> "ENDSTRUCT"
<STRUCT_STATEMENT>
| "TYPEDEF" "STRUCT" <USER_DATA_TYPE> "\n" <STRUCT_VARIABLE>
<USER_DATA_TYPE> "ENDSTRUCT"
| "TYPEDEF" "STRUCT" <USER_DATA_TYPE> "\n" <STRUCT_VARIABLE>
<USER_DATA_TYPE> "ENDSTRUCT" <STRUCT_STATEMENT>
```

Επίσης, σύμφωνα με την εκφώνηση, στη δήλωση των μεταβλητών εντός της δήλωσης δομής (δηλαδή εντός του STRUCT block), σε κάθε δήλωση μεταβλητής υπάρχει μόνο ένα όνομα μεταβλητής και δεν υπάρχει η δυνατότητα να υπάρξει λίστα με τα ονόματα των μεταβλητών όπως γίνεται οπουδήποτε αλλού στο πρόγραμμα σύμφωνα με τις προδιαγραφές της γλώσσας. Για αυτό το λόγο, δημιουργήθηκε ένας νέος κανόνας που ορίζει τη σύνταξη της δήλωσης των μεταβλητών με μόνο ένα όνομα μεταβλητής εντός της δήλωσης τύπου δεδομένων χρήστη ή της δήλωσης δομής. Ο νέος κανόνας παρουσιάζεται σε μορφή BNF παρακάτω:

```
<STRUCT_VARIABLE> ::= "VARS" <DATATYPE> <NAME> ";" "\n"
| "VARS" <DATATYPE> <NAME> <ARRAY> ";" "\n"
| "VARS" <DATATYPE> <NAME> ";" "\n" <STRUCT_VARIABLE>
| "VARS" <DATATYPE> <NAME> <ARRAY> ";" "\n" <STRUCT_VARIABLE>
```

ΣΧΕΔΙΑΣΤΙΚΗ ΠΑΡΑΔΟΧΗ:

Σύμφωνα με την εκφώνηση, πρέπει να γίνει η κατάλληλη μετατροπή ώστε να υποστηρίζονται οι τύποι δεδομένων χρήστη που ορίζονται από τη δομή TYPEDEF STRUCT, στη δήλωση μεταβλητών οπουδήποτε αλλού στο πρόγραμμα. Ωστόσο, χάριν ευκολίας υπάρχει η δυνατότητα υποστήριξης μόνο ενός τύπου δεδομένων χρήστη. **Στο παράδειγμα της τρέχουσας υλοποίησης, δίνεται η δυνατότητα στο χρήστη να ορίσει έναν τύπο δεδομένων ονόματι "Node", ο οποίος αποτελεί έναν κόμβο μιας linked list.** Συνεπώς γίνονται οι παρακάτω μετατροπές:

Δημιουργείται ο παρακάτω κανόνας:

<USER_DATA_TYPE> ::= "Node"

Πλέον, οι κανόνες <VARIABLE_DECLARATION> και <STRUCT_VARIABLE> ενημερώνονται ως εξής:

**<VARIABLE_DECLARATION> ::= "VARS" <DATATYPE_STATEMENT> <VARIABLE> ";"
"\n"**
| "VARS" <DATATYPE_STATEMENT> <VARIABLE> ";" "\n" <VARIABLE_DECLARATION>

<STRUCT_VARIABLE> ::= "VARS" <DATATYPE_STATEMENT> <NAME> ";" "\n"
| "VARS" <DATATYPE_STATEMENT> <NAME> <ARRAY> ";" "\n"
| "VARS" <DATATYPE_STATEMENT> <NAME> ";" "\n" <STRUCT_VARIABLE>
| "VARS" <DATATYPE_STATEMENT> <NAME> <ARRAY> ";" "\n" <STRUCT_VARIABLE>

Και ορίζεται ο κανόνας <DATATYPE_STATEMENT> ως εξής:

<DATATYPE_STATEMENT> ::= <DATATYPE> | <USER_DATA_TYPE>

Οι γραμματικοί κανόνες <NAME>, <ARRAY> και <DATATYPE> έχουν οριστεί στην τεκμηρίωση του ερωτήματος 1.

Επίσης, στο συντακτικό ορισμό σε μορφή BNF του πρώτου ερωτήματος, γίνεται αλλαγή στον κανόνα <PROGRAM_STATEMENT>, ο οποίος παίρνει την παρακάτω μορφή:

**<PROGRAM_STATEMENT> ::= <PROGRAM_DECLARATION> <MAIN_STATEMENT>
"\n"**
| <PROGRAM_DECLARATION> <FUNCTION_STATEMENT> <MAIN_STATEMENT> "\n"
| <PROGRAM_DECLARATION> <STRUCT_STATEMENT> <MAIN_STATEMENT> "\n"
**| <PROGRAM_DECLARATION> <STRUCT_STATEMENT> <FUNCTION_STATEMENT>
<MAIN_STATEMENT> "\n"**

4.3.2 ΕΠΙΚΑΙΡΟΠΟΙΗΣΗ ΤΗΣ ΥΛΟΠΟΙΗΣΗΣ ΤΟΥ ΛΕΞΙΚΟΥ ΚΑΙ ΣΥΝΤΑΚΤΙΚΟΥ ΑΝΑΛΥΤΗ

ΕΝΗΜΕΡΩΣΗ ΛΕΞΙΚΟΥ ΑΝΑΛΥΤΗ:

Στον λεξικό αναλυτή που υλοποιείται με το εργαλείο Flex προστίθεται η δυνατότητα να αναγνωρίζει τις δεσμευμένες λέξεις "TYPEDEF", "STRUCT" και "ENDSTRUCT". Αυτό γίνεται με την προσθήκη ορισμένων κανόνων στο Flex για την αναγνώριση των tokens που αντιστοιχούν στις παραπάνω δεσμευμένες λέξεις. Επίσης, προστίθεται η κατάλληλη λειτουργικότητα για την αναγνώριση του token που αντιστοιχεί στον τύπο δεδομένων χρήστη που περιγράφεται στη σχεδιαστική παραδοχή του ερωτήματος 2.

TOKEN	ΠΕΡΙΓΡΑΦΗ
TYPEDEF	Ακολουθία χαρακτήρων: "TYPEDEF"
STRUCT	Ακολουθία χαρακτήρων: "STRUCT"
ENDSTRUCT	Ακολουθία χαρακτήρων: "ENDSTRUCT"
USERDATATYPE	Ακολουθία χαρακτήρων: "Node"

ΕΝΗΜΕΡΩΜΕΝΟΣ ΚΩΔΙΚΑΣ ΕΙΣΟΔΟΥ ΣΤΟ FLEX (ΑΡΧΕΙΟ ΕΙΣΟΔΟΥ c-like_lexer.l):

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

%option yylineno

underscore      "_"
digit           [0-9]
letter          [a-zA-Z]
chartype        "CHAR"
inttype         "INTEGER"
userdatatype     "Node"
comparative      ">" | "<" | "==" | "!="
char            "."
charsequence     {char}*
strliteral       \"{charsequence}\"
charliteral      \"{char}\"
linecomment      "%\"{charsequence}
datatype         {chartype}|{inttype}
basiccharacter   {underscore}|{letter}|{digit}
number           {digit}+|{digit}+\".\"{digit}+
```



```

basicstring      {basiccharacter}*
name             {letter}{basicstring}|{underscore}{basicstring}
array            "["{number}"]"

%%

[ \t] { yylval.str = strdup(yytext); ECHO; printf("%s",yytext);}
"\n"  { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
NEWLINE; }
";"   { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return ';' ; }
":"   { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return ':' ; }
","   { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return ',' ; }
"("   { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return '(' ; }
")"   { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return ')' ; }
"+"   { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return '+' ; }
"-"   { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return '-' ; }
"^"   { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return '^' ; }
"*"   { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return '*' ; }
"/"   { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return '/' ; }
"="   { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return '=' ; }
"AND" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return AND; }
"OR"  { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return OR; }
":="  { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
ASSIGN_OPERATOR; }
"PROGRAM" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
PROGRAM; }
"VARS" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return VARS;}
"TYPEDEF" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
TYPEDEF; }
"STRUCT" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
STRUCT; }
"ENDSTRUCT" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
ENDSTRUCT; }
"FUNCTION" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
FUNCTION; }
"END_FUNCTION" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext);
return END_FUNCTION; }
"RETURN" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
RETURN; }
"WHILE" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
WHILE; }
"ENDWHILE" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
ENDWHILE; }
"FOR" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return FOR; }
"TO" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return TO; }
"STEP" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return STEP;}

```

```
"ENDFOR" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
ENDFOR; }
"IF" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return IF; }
"THEN" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return THEN;}
"ELSEIF" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
ELSEIF; }
"ELSE" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return ELSE;
}
"ENDIF" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
ENDIF; }
"SWITCH" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
SWITCH; }
"CASE" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return CASE;}
"DEFAULT" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
DEFAULT; }
"ENDSWITCH" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
ENDSWITCH; }
"PRINT" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
PRINT; }
"BREAK" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
BREAK; }
"STARTMAIN" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
STARTMAIN; }
"ENDMAIN" { yylval.str = strdup(yytext); ECHO; printf("%s\n",yytext); return
ENDMAIN; }
{strliteral} { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
STRLITERAL; }
{charliteral} { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
CHARLITERAL; }
{linecomment} { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); }
{userdatatype} { yylval.str = strdup(yytext); ECHO; printf("%s",yytext);
return USERDATATYPE; }
{datatype} { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
DATATYPE; }
{name} { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return NAME;}
{number} { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
NUM;}
{array} { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
ARRAY; }
{comparative} { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
COMP_OPERATOR; }

%%
```

ΕΝΗΜΕΡΩΣΗ ΣΥΝΤΑΚΤΙΚΟΥ ΑΝΑΛΥΤΗ:

Στο συντακτικό αναλυτή που υλοποιείται με το εργαλείο Bison, προστίθενται οι νέοι κανόνες που περιγράφονται σε μορφή BNF παραπάνω στην τεκμηρίωση του ερωτήματος 2 και ενημερώνεται ο βασικός συντακτικός κανόνας ορισμού ενός προγράμματος στην ψευδογλώσσα.

ΚΑΝΟΝΑΣ	ΠΕΡΙΓΡΑΦΗ
struct_statement	Αντιστοιχεί στον κανόνα <STRUCT_STATEMENT> του ερωτήματος 2. Αποτελεί τον κανόνα για τη δήλωση τύπου δεδομένων χρήστη ή τη δήλωση δομής.
struct_variable	Αντιστοιχεί στον κανόνα <STRUCT_VARIABLE> του ερωτήματος 2. Αποτελεί τον κανόνα για τη δήλωση των μεταβλητών εντός της δήλωσης τύπου δεδομένων χρήστη / δήλωσης δομής.
datatype	Αντιστοιχεί στον κανόνα <DATATYPE_STATEMENT> του ερωτήματος 2. Αποτελεί τον κανόνα για τον ορισμό των τύπων δεδομένων.

Οι κανόνες **variable_declaration** και **struct_variable**, ενημερώνονται ώστε να υποστηρίζουν τους τύπους δεδομένων που αναγνωρίζονται από τον κανόνα **datatype** που προστέθηκε.

Επίσης, ο κανόνας **program** ενημερώνεται ώστε να υποστηρίζεται η δήλωση δομής εντός του προγράμματος, στη θέση που υποδεικνύουν οι προδιαγραφές της ψευδογλώσσας.

ΕΝΗΜΕΡΩΜΕΝΟΣ ΚΩΔΙΚΑΣ ΕΙΣΟΔΟΥ ΣΤΟ BISON

(ΑΡΧΕΙΟ ΕΙΣΟΔΟΥ c-like_parser.y):

```
%{  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
  
#define YYERROR_VERBOSE 1  
  
void yyerror(char *);  
extern FILE *yyin;  
extern FILE *yyout;  
extern int yylineno;  
int line = 0;
```

```
FILE *diagnostics;
char *progr;
%}

%union {
    char *str;
}

%token PROGRAM <str>NAME ARRAY NUM COMP_OPERATOR AND OR STRLITERAL CHARLITERAL
%token TYPEDEF STRUCT ENDSTRUCT
%token VARS DATATYPE USERDATATYPE FUNCTION END_FUNCTION RETURN
%token STARTMAIN ENDMAIN
%token WHILE ENDWHILE
%token FOR ASSIGN_OPERATOR TO STEP ENDFOR
%token IF THEN ELSEIF ELSE ENDIF
%token SWITCH CASE DEFAULT ENDSWITCH
%token PRINT
%token BREAK
%token NEWLINE
%left ','
%left '+' '-'
%left '*' '/'
%right '^'

%start program

%%

program: program_declaration main_statement newline { printf("Code Parsed
successfully!\n"); fprintf(diagnostics, "Code Parsed successfully!\n"); }
    | program_declaration function main_statement newline { printf("Code
Parsed successfully!\n"); fprintf(diagnostics, "Code Parsed successfully!\n");
}
    | program_declaration struct_statement main_statement newline {
printf("Code Parsed successfully!\n"); fprintf(diagnostics, "Code Parsed
successfully!\n"); }
    | program_declaration struct_statement function main_statement newline
{ printf("Code Parsed successfully!\n"); fprintf(diagnostics, "Code Parsed
successfully!\n"); }
    ;

program_declaration: PROGRAM NAME newline { progr = $2; }
    ;

struct_statement: STRUCT NAME newline struct_variable ENDSTRUCT
    | struct_statement STRUCT NAME newline struct_variable ENDSTRUCT
```

```
| TYPEDEF STRUCT NAME newline struct_variable NAME ENDSTRUCT
| struct_statement TYPEDEF STRUCT NAME newline struct_variable NAME ENDSTRUCT
;

struct_variable: VARS datatype NAME ';' newline
                | VARS datatype NAME ARRAY ';' newline
                | VARS datatype NAME ';' newline variable_declaration
                | VARS datatype NAME ARRAY ';' newline variable_declaration
                ;

main_statement: STARTMAIN commands ENDMAIN /*{ printf("MAIN Statement
found\n"); }*/
                | STARTMAIN variable_declaration commands ENDMAIN /*{
printf("MAIN Statement found\n"); }*/
                ;

variable_declaration: VARS datatype variable ';' newline /*{ printf("Variable
Statement found\n"); }*/
                    | variable_declaration VARS datatype variable ';' newline
/*{ printf("Variable Statement found\n"); }*/
                    ;

function: function_declaration commands function_end newline /*{
printf("Function Statement found\n"); }*/
        | function_declaration variable_declaration commands function_end
newline /*{ printf("Function Statement found\n"); }*/
        | function function_declaration commands newline function_end newline
/*{ printf("Function Statement found\n"); }*/
        | function function_declaration variable_declaration commands
function_end newline /*{ printf("Function Statement found\n"); }*/
        ;

function_declaration: FUNCTION function_name newline /*{ printf("Function
Declared\n"); }*/
                    ;

function_name: NAME '(' variable ')'
              ;

function_end: RETURN NUM END_FUNCTION
            | RETURN CHARLITERAL END_FUNCTION
            | RETURN variable END_FUNCTION
            ;

command: assignment
        | print_statement
```

```
| loop_statement
| break_command
| control_statement
;

commands: command newline
        | commands command newline
;

assignment: variable '=' expression ';' /*{ printf("Assignment statement
found\n"); }*/
        | assignment variable '=' expression ';' /*{ printf("Assignment
statement found\n"); }*/
;

loop_statement: while_statement
              | for_statement
              ;

break_command: BREAK ';' /*{ printf("Break command found\n"); }*/
;

control_statement: if_statement
                 | switch_statement
                 ;

while_statement: WHILE '(' condition ')' newline commands ENDWHILE /*{
printf("While Loop statement found\n"); }*/
;

condition: expression COMP_OPERATOR expression
        | expression AND expression
        | expression OR expression
        | '(' condition ')' COMP_OPERATOR '(' condition ')'
        | '(' condition ')' AND '(' condition ')'
        | '(' condition ')' OR '(' condition ')'
;

for_statement: FOR NAME ASSIGN_OPERATOR NUM TO NUM STEP NUM newline commands
ENDFOR /*{ printf("For Loop statement found\n"); }*/
;

if_statement: IF '(' condition ')' THEN newline commands ENDIF /*{ printf("If
statement found\n"); }*/
        | IF '(' condition ')' THEN newline commands else_if_statement
ENDIF /*{ printf("If statement found\n"); }*/
```

```
| IF '(' condition ')' THEN newline commands else_statement ENDIF
/*{ printf("If statement found\n"); }*/
| IF '(' condition ')' THEN newline commands else_if_statement
else_statement ENDIF /*{ printf("If statement found\n"); }*/
;

else_if_statement: ELSEIF '(' condition ')' newline commands
| else_if_statement ELSEIF '(' condition ')' newline commands
;

else_statement: ELSE newline commands
;

switch_statement: SWITCH '(' expression ')' newline case ENDSWITCH /*{
printf("Switch statement found\n"); }*/
| SWITCH '(' expression ')' newline case default ENDSWITCH /*{
printf("Switch statement found\n"); }*/
;

case: CASE '(' expression ')' ':' newline commands
| case CASE '(' expression ')' ':' newline commands
;

default: DEFAULT ':' newline commands
;

print_statement: PRINT '(' STRLITERAL ')' ';' /*{ printf("Print Statement
found\n"); }*/
| PRINT '(' STRLITERAL ',' variable ')' ';' /*{ printf("Print
Statement found\n"); }*/
;

expression: literal
| variable
| function_name
| expression '+' expression
| expression '-' expression
| expression '^' expression
| expression '*' expression
| expression '/' expression
| '(' expression ')'
;

variable: NAME
| NAME ARRAY
| variable ',' variable
```

```
        ;

datatype: DATATYPE
        | USERDATATYPE
        ;

literal: NUM
        | STRLITERAL
        | CHARLITERAL
        ;

newline: NEWLINE { line++; }
        ;

%%

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

int main ( int argc, char **argv )
{
    ++argv; --argc;
    if ( argc > 0 )
        yyin = fopen( argv[0], "r" );
    else
        yyin = stdin;
    yyout = fopen ( "output.c", "w" );
    printf("C-like parser, implemented by Christos-Panagiotis Mpalatsouras,
Student ID = 1054335\n");
    printf("Program source code from parser input is following: \n\n");
    diagnostics = fopen("diagnostics.txt", "w");
    fprintf(diagnostics, "**** START of Diagnostic Messages ****\n\n");
    yyparse ();
    printf("Program Name: %s\n", progr);
    fprintf(diagnostics, "Program: %s\n\n", progr);
    printf("Lines of code parsed: %i\n", line);
    fprintf(diagnostics, "Lines of code parsed: %i\n", line);
    fprintf(diagnostics, "\n**** END of Diagnostic Messages ****\n");
    fclose(diagnostics);

    return 0;
}
```


4.3.3 ΠΑΡΑΔΕΙΓΜΑ ΛΕΙΤΟΥΡΓΙΑΣ (TEST CASE):

Στο παράδειγμα λειτουργίας του ερωτήματος 1.b προστίθενται οι παρακάτω γραμμές κώδικα για τη δήλωση ενός τύπου δεδομένων χρήστη, με το όνομα «Node», ο οποίος αναπαριστά έναν κόμβο διασυνδεδεμένης λίστας. Επίσης, έχει ενημερωθεί ο αναλυτής, ώστε να αναγνωρίζονται οι μεταβλητές που θα δηλώνονται παρακάτω με αυτό τον τύπο δεδομένων.

ΠΑΡΑΔΕΙΓΜΑ ΔΗΛΩΣΗΣ ΤΥΠΟΥ ΔΕΔΟΜΕΝΩΝ ΑΠΟ ΤΟ ΧΡΗΣΤΗ:

```
TYPEDEF STRUCT Node
    VARS INTEGER value;
    VARS Node prev;
    VARS Node next;
Node ENDSTRUCT
```

ΣΥΝΟΛΙΚΟΣ ΚΩΔΙΚΑΣ ΠΑΡΑΔΕΙΓΜΑΤΟΣ ΣΤΗΝ ΨΕΥΔΟΓΛΩΣΣΑ:

```
PROGRAM Sorting_Algorithms %This is a test program for lexer and parser development
purposes
TYPEDEF STRUCT Node
    VARS INTEGER value;
    VARS Node prev;
    VARS Node next;
Node ENDSTRUCT FUNCTION BubbleSort(K[20], m)
    VARS INTEGER i,j,head,temp;
    i = 0;
    j = 0;
    head = m;
    WHILE (head > 1)
        j = 0;
        head2 = head - 1;
        PRINT("Sorting elements 0 --> %d, ", head2);
        FOR i:=0 TO 20 STEP 1
            IF (K_i > K_i_plus_1) THEN
                temp = K_i_plus_1;
                K_i_plus_1 = K_i;
                K_i = temp;
                j = j + 1;
            ENDIF
        ENDFOR
        PRINT("Element swaps in the current table scan: %d\n", j);
        IF (j == 0) THEN
            PRINT("No swap during last table scan. The table sorting completed
successfully\n");
            BREAK;
        ENDIF
        head = head - 1;
```

```
ENDWHILE
RETURN K_m END_FUNCTION
FUNCTION InsertionSort(K[20], m)
  VARS INTEGER i,j,k;
  i = 0;
  j = 0;
  k = 0;
  FOR j:=2 TO 20 STEP 1
    PRINT("Sorting elements 0 --> %d\n", j);
    k = K_j;
    i = j - 1;
    WHILE ((i > 0) AND (k < K_i))
      K_i_plus_1 = K_i;
      i = i - 1;
    ENDWHILE
    K_i_plus_1 = k;
  ENDFOR
RETURN K_m END_FUNCTION
STARTMAIN VARS INTEGER n,i,c,A[20];
  n = 20;
  A[20] = values;
  PRINT("Initial Table: \n");
  FOR i:=0 TO 20 STEP 1
    PRINT("%d ", A_i);
  ENDFOR
  PRINT("\nOptions: \n");
  PRINT("1. Bubble-Sort:\n");
  PRINT("2. Insertion-Sort:\n");
  PRINT("3. Selection-Sort:\n");
  PRINT("Select Algorithm >> "); %scanf an option from keyboard
  SWITCH (option)
  CASE (1):
    A_n = BubbleSort(A,n);
    PRINT("Results: \n");
    FOR i:=0 TO 20 STEP 1
      PRINT("%d ", A_i);
    ENDFOR
    PRINT("\n");
  CASE (2):
    A_n = InsertionSort(A,n);
    PRINT("Results: \n");
    FOR i:=0 TO 20 STEP 1
      PRINT("%d ", A_i);
    ENDFOR
    PRINT("\n");
  DEFAULT:
    PRINT("No option selected\n");
  ENDSWITCH
ENDMAIN
```

ΠΑΡΑΔΕΙΓΜΑ ΕΠΙΤΥΧΟΥΣ ΕΚΤΕΛΕΣΗΣ:

Δίνοντας ως είσοδο τον παραπάνω δοκιμαστικό κώδικα στον αναλυτή που υλοποιήθηκε στα πλαίσια της εργασίας, προκύπτει το παρακάτω αποτέλεσμα στα στιγμιότυπα οθόνης που ακολουθούν:

```
takis@takis-VirtualBox: ~/parser
takis@takis-VirtualBox:~/parser$ bison -y -d c-like_parser.y
takis@takis-VirtualBox:~/parser$ flex c-like_lexer.l
takis@takis-VirtualBox:~/parser$ gcc y.tab.c lex.yy.c -lfl
y.tab.c: in function 'yyparse':
y.tab.c:1477:16: warning: implicit declaration of function 'yylex' [-Wimplicit-function-declaration]
1477 |         yychar = yylex ();
      |         ^~~~~~
y.tab.c:1667:18: warning: passing argument 1 of 'yyerror' discards 'const' qualifier from pointer target type [-Wdiscarded-qualifiers]
1667 |         yyerror (ymsgp);
      |         ^~~~~~
c-like_parser.y:9:14: note: expected 'char *' but argument is of type 'const char *'
9 | void yyerror(char *);
  |               ^~~~~
takis@takis-VirtualBox:~/parser$ ./a.out prog.c
C-like parser, implemented by Christos-Panagiotis Mpalatsouras, Student ID = 1054335
Program source code from parser input is following:

PROGRAM Sorting Algorithms %This is a test program for lexer and parser development purposes
TYPEDEF STRUCT Node
  VARS INTEGER value;
  VARS Node prev;
  VARS Node next;
Node ENDSTRUCT FUNCTION BubbleSort(K[20], n)
  VARS INTEGER i,j,head,temp;
  i = 0;
  j = 0;
  head = n;
  WHILE (head > 1)
    j = 0;
    head2 = head - 1;
    PRINT("Sorting elements 0 --> %d, ", head2);
    FOR i:=0 TO 20 STEP 1
      IF (K_i > K_i_plus_1) THEN
        temp = K_i_plus_1;
        K_i_plus_1 = K_i;
        K_i = temp;
        j = j + 1;
      ENDIF
    ENDFOR
    PRINT("Element swaps in the current table scan: %d\n", j);
    IF (j == 0) THEN
      PRINT("No swap during last table scan. The table sorting completed successfully\n");
      BREAK;
    ENDIF
    head = head - 1;
  ENDWHILE
RETURN K_n END_FUNCTION
FUNCTION InsertionSort(K[20], n)
  VARS INTEGER i,j,k;
  i = 0;
```

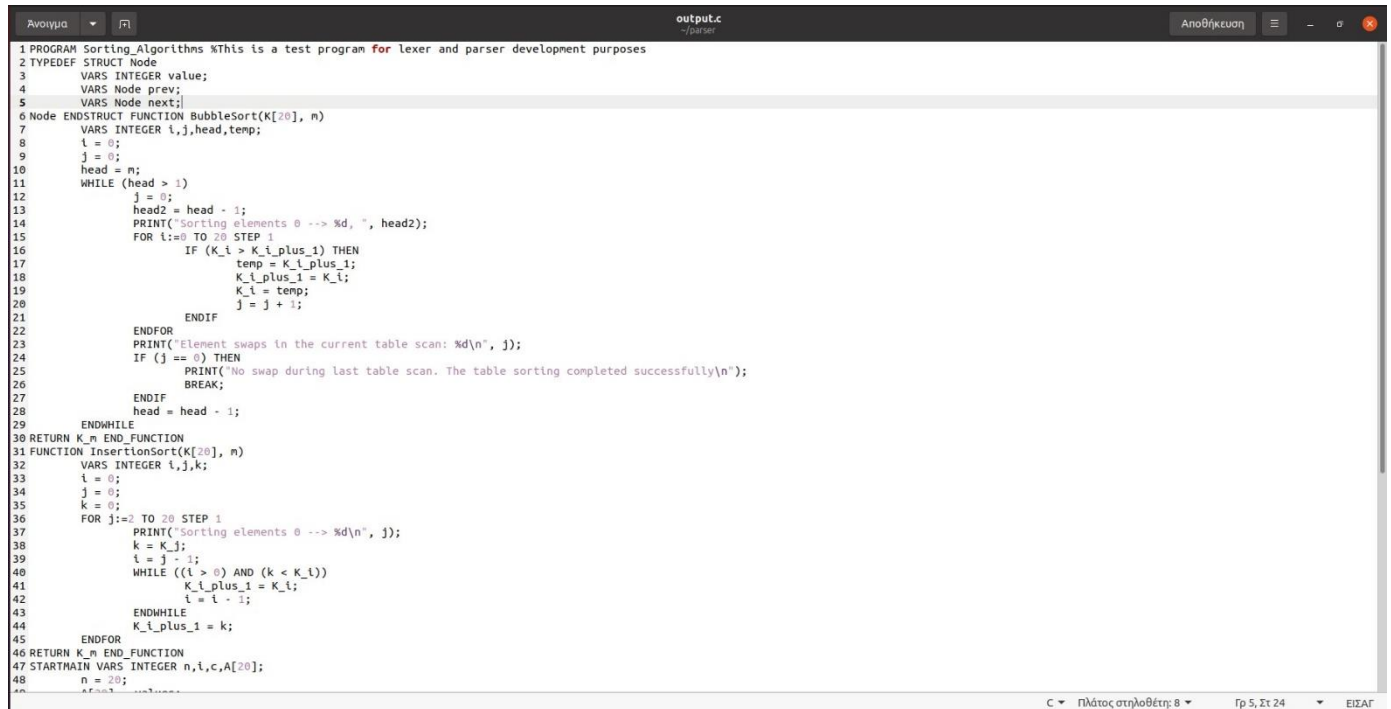
```

  i = 0;
  j = 0;
  k = 0;
  FOR j:=2 TO 20 STEP 1
    PRINT("Sorting elements 0 --> %d\n", j);
    k = K_j;
    i = j - 1;
    WHILE ((i > 0) AND (k < K_i))
      K_i_plus_1 = K_i;
      i = i - 1;
    ENDWHILE
    K_i_plus_1 = k;
  ENDFOR
RETURN K_n END_FUNCTION
STARTMAIN VARS INTEGER n,t,c,A[20];
n = 20;
A[20] = values;
PRINT("Initial Table: \n");
FOR i:=0 TO 20 STEP 1
  PRINT("%d ", A_i);
ENDFOR
PRINT("\nOptions: \n");
PRINT("1. Bubble-Sort:\n");
PRINT("2. Insertion-Sort:\n");
PRINT("3. Selection-Sort:\n");
PRINT("Select Algorithm >> "); %scanf an option from keyboard
SWITCH (option)
CASE (1):
  A_n = BubbleSort(A,n);
  PRINT("Results: \n");
  FOR i:=0 TO 20 STEP 1
    PRINT("%d ", A_i);
  ENDFOR
  PRINT("\n");
CASE (2):
  A_n = InsertionSort(A,n);
  PRINT("Results: \n");
  FOR i:=0 TO 20 STEP 1
    PRINT("%d ", A_i);
  ENDFOR
  PRINT("\n");
DEFAULT:
  PRINT("No option selected\n");
ENDSWITCH
ENDMAIN

Code Parsed successfully!
Program Name: Sorting Algorithms
Lines of code parsed: 77
takis@takis-VirtualBox:~/parser$
```

Επειδή το παραπάνω αποτέλεσμα της εκτέλεσης του αναλυτή δεν χώραγε σε μια οθόνη, παραπάνω παρουσιάζονται δύο στιγμιότυπα οθόνης όπου το δεύτερο αποτελεί συνέχεια του πρώτου.

Ο κώδικας της εισόδου, έχει επιστραφεί στην έξοδο στο παραπάνω τερματικό και στο αρχείο output.c:



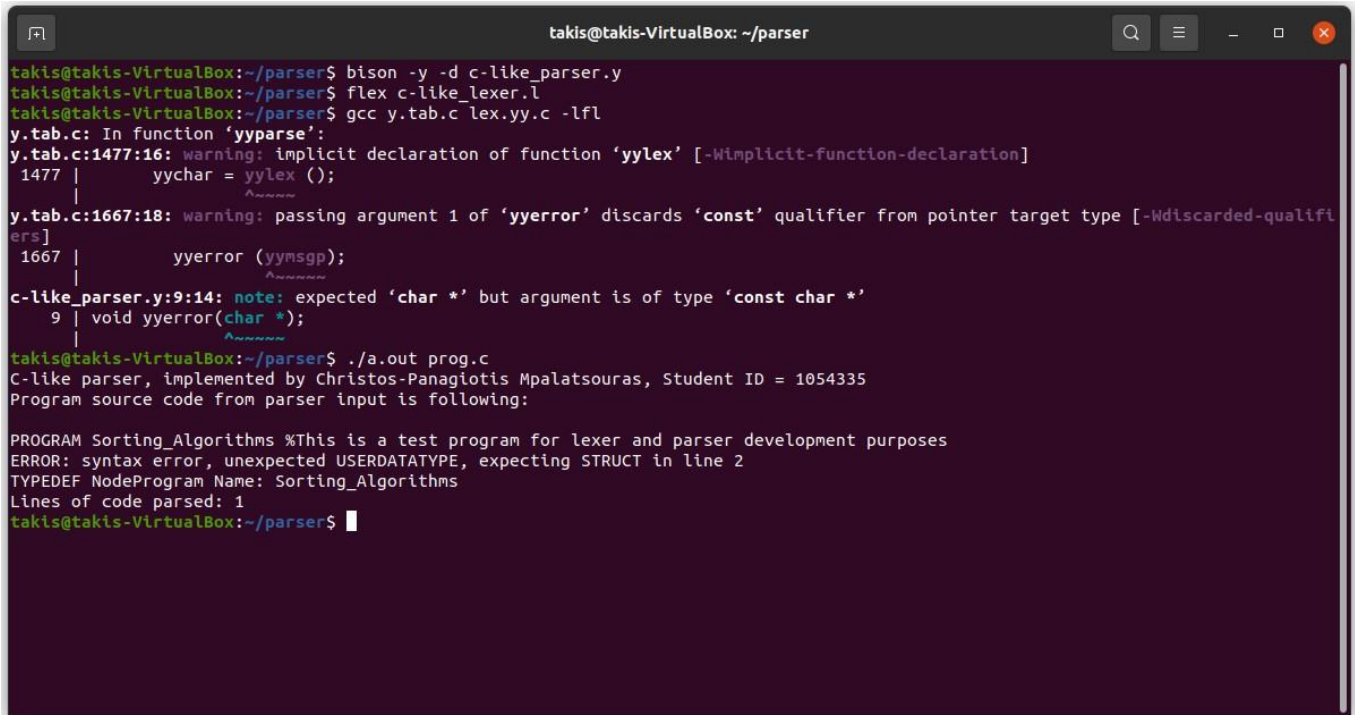
```
1 PROGRAM Sorting_Algorithms %This is a test program for lexer and parser development purposes
2 TYPEDEF STRUCT Node
3   VARS INTEGER value;
4   VARS Node prev;
5   VARS Node next;
6 Node ENDSTRUCT FUNCTION BubbleSort(K[20], n)
7   VARS INTEGER i,j,head,temp;
8   i = 0;
9   j = 0;
10  head = n;
11  WHILE (head > 1)
12    j = 0;
13    head2 = head - 1;
14    PRINT('Sorting elements 0 --> %d, ', head2);
15    FOR i:=0 TO 20 STEP 1
16      IF (K_i > K_i_plus_1) THEN
17        temp = K_i_plus_1;
18        K_i_plus_1 = K_i;
19        K_i = temp;
20        j = j + 1;
21      ENDIF
22    ENDFOR
23    PRINT('Element swaps in the current table scan: %d\n', j);
24    IF (j == 0) THEN
25      PRINT('No swap during last table scan. The table sorting completed successfully\n');
26      BREAK;
27    ENDIF
28    head = head - 1;
29  ENDWHILE
30 RETURN K_m END_FUNCTION
31 FUNCTION InsertionSort(K[20], n)
32   VARS INTEGER i,j,k;
33   i = 0;
34   j = 0;
35   k = 0;
36   FOR j:=2 TO 20 STEP 1
37     PRINT('Sorting elements 0 --> %d\n', j);
38     k = K_j;
39     i = j - 1;
40     WHILE ((i > 0) AND (k < K_i))
41       K_i_plus_1 = K_i;
42       i = i - 1;
43     ENDWHILE
44     K_i_plus_1 = k;
45   ENDFOR
46 RETURN K_m END_FUNCTION
47 STARTMAIN VARS INTEGER n,i,c,A[20];
48   n = 20;
```

Ακολουθεί στην επόμενη σελίδα το παράδειγμα ανεπιτυχούς εκτέλεσης.

ΠΑΡΑΔΕΙΓΜΑ ΑΝΕΠΙΤΥΧΟΥΣ ΕΚΤΕΛΕΣΗΣ:

Έστω ότι από τη γραμμή 2 λείπει η λέξη «STRUCT».

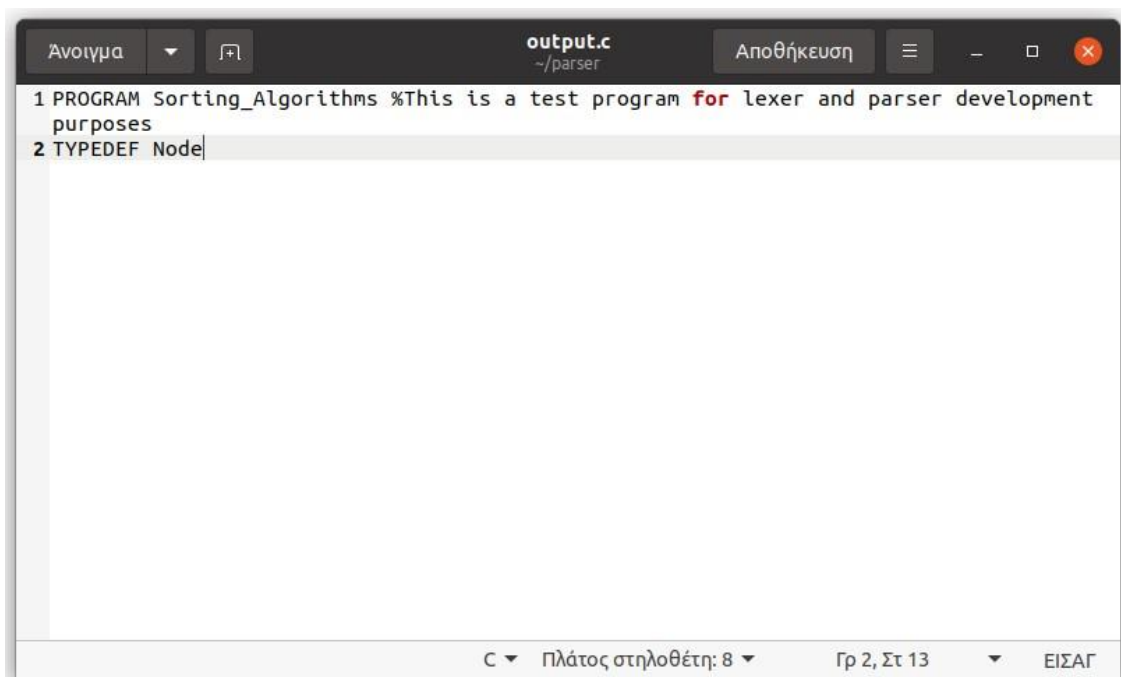
Εμφανίζεται το αντίστοιχο μήνυμα σφάλματος στο τερματικό, όπως φαίνεται στο παρακάτω στιγμιότυπο οθόνης:



```
takis@takis-VirtualBox: ~/parser
takis@takis-VirtualBox:~/parser$ bison -y -d c-like_parser.y
takis@takis-VirtualBox:~/parser$ flex c-like_lexer.l
takis@takis-VirtualBox:~/parser$ gcc y.tab.c lex.yy.c -lfl
y.tab.c: In function 'yyparse':
y.tab.c:1477:16: warning: implicit declaration of function 'yylex' [-Wimplicit-function-declaration]
1477 |         yychar = yylex ();
      |
y.tab.c:1667:18: warning: passing argument 1 of 'yyerror' discards 'const' qualifier from pointer target type [-Wdiscarded-qualifiers]
1667 |         yyerror (ymsgp);
      |
c-like_parser.y:9:14: note: expected 'char *' but argument is of type 'const char *'
9 | void yyerror(char *);
  |
takis@takis-VirtualBox:~/parser$ ./a.out prog.c
C-like parser, implemented by Christos-Panagiotis Mpalatsouras, Student ID = 1054335
Program source code from parser input is following:

PROGRAM Sorting_Algorithms %This is a test program for lexer and parser development purposes
ERROR: syntax error, unexpected USERDATATYPE, expecting STRUCT in line 2
TYPEDEF NodeProgram Name: Sorting_Algorithms
Lines of code parsed: 1
takis@takis-VirtualBox:~/parser$
```

Ο κώδικας της εισόδου δεν έχει επιστραφεί ολόκληρος στην έξοδο, στο παραπάνω τερματικό και στο αρχείο output.c, καθώς έχει σταματήσει στη γραμμή που εμφανίζεται το σφάλμα.



```
Ανοίγμα  output.c  Αποθήκευση
~/parser
1 PROGRAM Sorting_Algorithms %This is a test program for lexer and parser development
  purposes
2 TYPEDEF Node
```

C ▾ Πλάτος στηλοθέτη: 8 ▾ Γρ 2, Στ 13 ▾ ΕΙΣΑΓ

4.4 ΕΡΩΤΗΜΑ 3 – ΕΛΕΓΧΟΣ ΣΩΣΤΗΣ ΔΗΛΩΣΗΣ ΜΕΤΑΒΛΗΤΩΝ ΚΑΙ ΣΥΝΑΡΤΗΣΕΩΝ

Σε αυτή την ενότητα, τροποποιείται ο συντακτικός αναλυτής ώστε να υποστηρίζει τη δυνατότητα ελέγχου σωστής δήλωσης των μεταβλητών, ώστε όταν χρησιμοποιούνται μεταβλητές οπουδήποτε στο πρόγραμμα να ελέγχεται αν έχουν δηλωθεί προηγουμένως. Ομοίως, ο αναλυτής τροποποιείται ώστε να ελέγχει εάν οι συναρτήσεις που χρησιμοποιούνται στο πρόγραμμα έχουν δηλωθεί προηγουμένως.

ΣΥΝΟΠΤΙΚΗ ΤΕΚΜΗΡΙΩΣΗ ΤΗΣ ΥΛΟΠΟΙΗΣΗΣ:

Αρχικά, για να υλοποιηθεί ο έλεγχος δήλωσης μεταβλητών και συναρτήσεων, πρέπει να υπάρχουν δύο λίστες στις οποίες θα αποθηκεύονται τα ονόματα των μεταβλητών και των συναρτήσεων κατά τη δήλωση. Επίσης, όταν γίνονται reduce οι κανόνες που κάνουν χρήση μεταβλητών ή συναρτήσεων (για παράδειγμα: κανόνες σύνταξης εντολών ανάθεσης), θα πρέπει να γίνεται διάτρεξη της αντίστοιχης λίστας ώστε να ελεγχθεί εάν υπάρχει σε αυτή το ζητούμενο όνομα μεταβλητής ή συνάρτησης που χρησιμοποιείται από την εντολή. Εάν το ζητούμενο όνομα μεταβλητής ή συνάρτησης δεν υπάρχει θα πρέπει η ανάλυση να διακόπτεται με την εμφάνιση του αντίστοιχου σφάλματος.

4.4.1 ΕΝΗΜΕΡΩΣΗ ΤΟΥ ΣΥΝΤΑΚΤΙΚΟΥ ΟΡΙΣΜΟΥ ΤΗΣ ΓΛΩΣΣΑΣ

Για να μπορέσουν να προστεθούν οι νέες λειτουργίες, γίνονται αλλαγές στους παρακάτω κανόνες του συντακτικού της γλώσσας.

Αρχικά, παρατηρήθηκε ότι στη γραμματική της γλώσσας που υλοποιήθηκε στα προηγούμενα ερωτήματα, ο κανόνας σύνταξης των ονομάτων των μεταβλητών <VARIABLE> του συντακτικού χρησιμοποιήθηκε και σε κανόνες σύνταξης για εντολές δήλωσης μεταβλητών και σε κανόνες σύνταξης εντολές που χρησιμοποιούν μεταβλητές. Συνεπώς για τους σκοπούς της υλοποίησης αυτού του ερωτήματος υλοποιήθηκε ένας επιπλέον κανόνας σύνταξης των ονομάτων των μεταβλητών, όπου χρησιμοποιείται στον κανόνα σύνταξης των εντολών δήλωσης μεταβλητών και ο άλλος κανόνας που προϋπήρχε χρησιμοποιείται στις εντολές που χρησιμοποιούν μεταβλητές. Αυτό γίνεται διότι όταν καλείται ο κανόνας της δήλωσης μεταβλητών, θα κληθεί ο κανόνας <DECLARED_VARIABLE> ο οποίος θα περιέχει ένα action που θα αποθηκεύσει τα ονόματα των μεταβλητών που δηλώνονται στη λίστα. Ενώ όταν καλούνται οι εντολές που χρησιμοποιούν μεταβλητές, θα κληθεί ο κανόνας <VARIABLE> που θα

περιέχει ένα action που θα ψάχνει στη λίστα το όνομα της μεταβλητής που χρησιμοποιείται, ώστε να ελεγχθεί αν η μεταβλητή έχει δηλωθεί.

Επίσης, για τη λίστα των ορισμάτων όταν δηλώνεται μια συνάρτηση, υλοποιείται ένας νέος κανόνας <FUNCTION_ARGUMENTS>, ο οποίος ορίζει τη σύνταξη των ονομάτων των ορισμάτων της συνάρτησης χωρισμένα με κόμμα. Όταν γίνεται reduce αυτός ο κανόνας σύνταξης των ορισμάτων μίας συνάρτησης κατά τη δήλωσή της, ακολουθείται η ίδια διαδικασία με τη δήλωση μεταβλητών, ώστε τα ορίσματα μιας νέας συνάρτησης που δηλώνεται να δηλωθούν και αυτά ως μεταβλητές.

Σχετικά με τη χρήση συναρτήσεων σε εντολές ανάθεσης, δημιουργείται ένας νέος κανόνας <FUNCTION_STATEMENT> ο οποίος για τα ορίσματα της συνάρτησης, χρησιμοποιεί τον κανόνα <VARIABLE> ο οποίος μόλις γίνει reduce θα κάνει έλεγχο δήλωσης των μεταβλητών που αποτελούν τα ορίσματα της συνάρτησης.

Επιπρόσθετα, κατά τη δήλωση μεταβλητών εντός της δήλωσης δομής ή τύπου δεδομένων χρήστη, ακολουθείται η ίδια διαδικασία με τη δήλωση μεταβλητών, ώστε οι μεταβλητές του struct να αποθηκευτούν στη λίστα με τα ονόματα των μεταβλητών.

Ο συντακτικός ορισμός της γλώσσας ενημερώνεται ως εξής:

Δήλωση Μεταβλητών:

```
<VARIABLE_DECLARATION> ::= "VARS" <DATATYPE> <DECLARED_VARIABLE> ";"  
"\n"  
| "VARS" <DATATYPE> <VARIABLE> ";" "\n" <VARIABLE_DECLARATION>
```

Ο νέος κανόνας για τη σύνταξη των ονομάτων των δηλωμένων μεταβλητών ορίζεται ως εξής:

```
<DECLARED_VARIABLE> ::= <NAME> | <NAME><ARRAY> | <DECLARED_VARIABLE>  
", " <DECLARED_VARIABLE>
```

Δήλωση Συναρτήσεων:

Από τους κανόνες που αφορούν τη δήλωση συναρτήσεων, ενημερώνεται ο κανόνας για τη σύνταξη του ονόματος μιας συνάρτησης:

```
<FUNCTION_NAME> ::= <NAME> "(" <FUNCTION_ARGUMENTS> ")"  
<FUNCTION_ARGUMENTS> ::= <NAME> | <FUNCTION_ARGUMENTS> ", "  
<FUNCTION_ARGUMENTS>
```

Χρήση Συναρτήσεων:

<FUNCTION_STATEMENT> ::= <NAME> "(" < VARIABLE> ")"

Ενημέρωση του κανόνα <EXPRESSION>:

**<EXPRESSION> ::= <NUMBER>
| <VARIABLE>
| <FUNCTION_STATEMENT>
| <EXPRESSION> "+" <EXPRESSION>
| <EXPRESSION> "-" <EXPRESSION>
| <EXPRESSION> "^" <EXPRESSION>
| <EXPRESSION> "*" <EXPRESSION>
| <EXPRESSION> "/" <EXPRESSION>
| "(" <EXPRESSION> ")"**

ΣΧΕΔΙΑΣΤΙΚΗ ΠΑΡΑΔΟΧΗ:

Σε περίπτωση που κάποιο όρισμα (argument) μιας συνάρτησης είναι πίνακας, στη λίστα με τα ορίσματα της συνάρτησης κατά τον ορισμό ή την κλήση της, θα πρέπει να υπάρχει μόνο το όνομα του πίνακα και όχι οι διαστάσεις του.

4.4.2 ΤΡΟΠΟΠΟΙΗΣΗ ΤΟΥ ΣΥΝΤΑΚΤΙΚΟΥ ΑΝΑΛΥΤΗ (PARSER)

ΠΡΟΣΘΗΚΗ ΛΕΙΤΟΥΡΓΙΚΟΤΗΤΑΣ ΕΛΕΓΧΟΥ ΔΗΛΩΣΗΣ ΜΕΤΑΒΛΗΤΩΝ ΚΑΙ ΣΥΝΑΡΤΗΣΕΩΝ:

Για την αποθήκευση των ονομάτων των μεταβλητών και των συναρτήσεων, υλοποιούνται δύο διασυνδεδεμένες λίστες, μια για τα ονόματα μεταβλητών και μία για τα ονόματα των συναρτήσεων.

Αρχικά, υλοποιούνται οι δομές δεδομένων για κάθε κόμβο που ανήκει στην αντίστοιχη λίστα. Η δομή struct variableEntity αποτελεί κάθε κόμβο της λίστας με τα ονόματα των μεταβλητών, ενώ η δομή struct functionEntity αποτελεί κάθε κόμβο της λίστας με τα ονόματα των συναρτήσεων. Κάθε κόμβος έχει μια μεταβλητή char * Name για το όνομα και ένα δείκτη προς τον επόμενο κόμβο της λίστας.

Στη συνέχεια, οι συναρτήσεις void printVariableList και void printFunctionList εκτυπώνουν τα ονόματα για όλα τα στοιχεία της εκάστοτε διασυνδεδεμένης λίστας.

Επιπρόσθετα, οι συναρτήσεις void variableListAppend και void functionListAppend χρησιμοποιούνται για την προσθήκη ενός νέου κόμβου στο τέλος της κάθε λίστας. Εάν η λίστα είναι κενή, ο νέος κόμβος είναι ο πρώτος κόμβος της λίστας.

Επίσης, οι συναρτήσεις `int variableSearch` και `int functionSearch` κάνουν αναζήτηση ενός ονόματος μεταβλητής ή συνάρτησης αντίστοιχα. Πιο συγκεκριμένα, διατρέχουν την εκάστοτε λίστα και εάν βρεθεί ο κόμβος με το ζητούμενο όνομα, επιστρέφεται η τιμή 1, σε διαφορετική περίπτωση επιστρέφεται μηδέν.

Τέλος, δηλώνονται οι δείκτες προς το πρώτο αντικείμενο της κάθε λίστας και αρχικοποιούνται με την τιμή `NULL`. Η λίστα με τα ονόματα των μεταβλητών ονομάζεται `variables` και η λίστα με τα ονόματα των συναρτήσεων ονομάζεται `functions`.

Δήλωση μεταβλητών έχουμε κάθε φορά που καλούνται οι εξής κανόνες του κώδικα εισόδου του Bison: **`declared_variables`** που καλείται από τον κανόνα `variable_declaration`, **`struct_variable`** που αφορά τη δήλωση μεταβλητών μέσα σε `struct` και **`function_name`** για να δηλωθούν οι μεταβλητές που αποτελούν τα ορίσματα μιας συνάρτησης που δηλώνεται. Συνεπώς, κάθε φορά που καλείται ένας από τους παραπάνω κανόνες, στο `action` υπάρχουν οι εντολές ώστε να γίνεται έλεγχος αν το όνομα της μεταβλητής που δηλώνεται υπάρχει στη λίστα με τη συνάρτηση `variableSearch` και αν δεν υπάρχει στη λίστα, γίνεται προσθήκη του νέου ονόματος μεταβλητής στη λίστα `variables`. Με αυτό τον τρόπο αποφεύγονται και οι διπλότυπες εγγραφές στη λίστα `variables`.

Χρήση μεταβλητών έχουμε όταν γίνονται `reduce` οι κανόνες του κώδικα εισόδου του Bison: **`variable`**, **`for_statement`** και **`function_statement`**. Στο `action` αυτών των κανόνων υπάρχουν οι εντολές για τον έλεγχο της σωστής δήλωσης των μεταβλητών που χρησιμοποιούνται από τις εντολές του προγράμματος που χρησιμοποιούν μεταβλητές, ώστε να γίνεται έλεγχος με τη συνάρτηση `variableSearch` εάν το όνομα της μεταβλητής που χρησιμοποιήθηκε υπάρχει στη λίστα `variables`. Εάν το όνομα της μεταβλητής που χρησιμοποιήθηκε δεν υπάρχει, εκτυπώνεται το μήνυμα σφάλματος που περιλαμβάνει και τη γραμμή που εντοπίζεται το σφάλμα, ενώ η συντακτική ανάλυση διακόπτεται με τη χρήση της «`YYABORT;`».

Ομοίως για τις συναρτήσεις, ο κανόνας του Bison που δηλώνει μια συνάρτηση, είναι ο **`function_name`**. Όταν κληθεί αυτός ο κανόνας, θα εκτελεστεί το `action` αυτού του κανόνα, το οποίο με τη χρήση της συνάρτησης `functionSearch` ελέγχεται αν υπάρχει στη λίστα `functions` το όνομα της συνάρτησης που δηλώνεται. Εάν το όνομα δεν υπάρχει, προστίθεται στη λίστα με τη συνάρτηση `functionListAppend`. Για τη χρήση των συναρτήσεων από τις εντολές του προγράμματος, ο κανόνας του Bison που ενεργοποιείται κατά τη χρήση μιας συνάρτησης είναι ο **`function_statement`**. Στο `action` αυτού του κανόνα υπάρχουν οι εντολές στις οποίες γίνεται έλεγχος με τη συνάρτηση `functionSearch` αν το όνομα της συνάρτησης υπάρχει στη λίστα `functions` και αν δεν

υπάρχει, εκτυπώνεται το αντίστοιχο μήνυμα σφάλματος με τη γραμμή στην οποία βρίσκεται το σφάλμα, ενώ η συντακτική ανάλυση διακόπτεται με τη χρήση της «YYABORT;».

Στη συνάρτηση main() του κώδικα εισόδου του Bison, γίνεται κλήση των συναρτήσεων printVariableList και printFunctionList, ώστε στο τέλος της εκτέλεσης του αναλυτή να εκτυπώνονται όλα τα ονόματα μεταβλητών και συναρτήσεων που έχουν δηλωθεί.

ΕΝΗΜΕΡΩΜΕΝΟΣ ΚΩΔΙΚΑΣ ΕΙΣΟΔΟΥ ΣΤΟ BISON (ΑΡΧΕΙΟ ΕΙΣΟΔΟΥ c-like_parser.y):

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

#define YYERROR_VERBOSE 1

void yyerror(char *);
extern FILE *yyin;
extern FILE *yyout;
extern int yylineno;
int line = 0;
FILE *diagnostics;
char *progr;

struct variableEntity
{
    char *Name;
    struct variableEntity *next;
};
struct functionEntity
{
    char *Name;
    struct functionEntity *next;
};
void printVariableList(struct variableEntity *node)
{
    while(node != NULL)
    {
        printf("Variable Name: %s\n", node->Name);
        node = node->next;
    }
}
void printFunctionList(struct functionEntity *node)
{

```

```
while(node != NULL)
{
    printf("Function Name: %s\n", node->Name);
    node = node->next;
}
}
void variableListAppend(struct variableEntity** head_rf, char *name)
{
    struct variableEntity* new_var = (struct variableEntity*)
malloc(sizeof(struct variableEntity));
    struct variableEntity *end = *head_rf;

    new_var->Name = name;
    new_var->next = NULL;

    if(*head_rf == NULL)
    {
        *head_rf = new_var;
        return;
    }

    while(end->next != NULL)
    {
        end = end->next;
    }

    end->next = new_var;

    return;
}
void functionListAppend(struct functionEntity** head_rf, char *name)
{
    struct functionEntity* new_var = (struct functionEntity*)
malloc(sizeof(struct functionEntity));
    struct functionEntity *end = *head_rf;

    new_var->Name = name;
    new_var->next = NULL;

    if(*head_rf == NULL)
    {
        *head_rf = new_var;
        return;
    }

    while(end->next != NULL)
```

```
{
    end = end->next;
}

end->next = new_var;

return;
}
int variableSearch(struct variableEntity* head, char *key)
{
    struct variableEntity* lookup = head;

    while(lookup != NULL)
    {
        if(strcmp(lookup->Name, key) == 0)
        {
            return 1;
        }
        lookup = lookup->next;
    }
    return 0;
}
int functionSearch(struct functionEntity* head, char *key)
{
    struct functionEntity* lookup = head;

    while(lookup != NULL)
    {
        if(strcmp(lookup->Name, key) == 0)
        {
            return 1;
        }
        lookup = lookup->next;
    }
    return 0;
}

struct variableEntity* variables = NULL;
struct functionEntity* functions = NULL;

%}

%union {
    char *str;
}
```

```
%token PROGRAM <str>NAME ARRAY NUM COMP_OPERATOR AND OR STRLITERAL CHARLITERAL
%token TYPEDEF STRUCT ENDSTRUCT
%token VARS DATATYPE USERDATATYPE FUNCTION END_FUNCTION RETURN
%token STARTMAIN ENDMAIN
%token WHILE ENDWHILE
%token FOR ASSIGN_OPERATOR TO STEP ENDFOR
%token IF THEN ELSEIF ELSE ENDIF
%token SWITCH CASE DEFAULT ENDSWITCH
%token PRINT
%token BREAK
%token NEWLINE
%left ','
%left '+' '-'
%left '*' '/'
%right '^'

%start program

%%

program: program_declaration main_statement newline { printf("Code Parsed
successfully!\n"); fprintf(diagnostics, "Code Parsed successfully!\n"); }
    | program_declaration function main_statement newline { printf("Code
Parsed successfully!\n"); fprintf(diagnostics, "Code Parsed successfully!\n");
}
    | program_declaration struct_statement main_statement newline {
printf("Code Parsed successfully!\n"); fprintf(diagnostics, "Code Parsed
successfully!\n"); }
    | program_declaration struct_statement function main_statement newline
{ printf("Code Parsed successfully!\n"); fprintf(diagnostics, "Code Parsed
successfully!\n"); }
    ;

program_declaration: PROGRAM NAME newline { progr = $2; }
    ;

variable_declaration: VARS datatype declared_variables ';' newline
    | variable_declaration VARS datatype declared_variables
    ';' newline
    ;

declared_variables: NAME { if(variableSearch(variables, $1) == 0){
variableListAppend(&variables, $1); } }
    | NAME ARRAY { if(variableSearch(variables, $1) ==
0){variableListAppend(&variables, $1); } }
    | declared_variables ',' declared_variables
```

```
        ;

struct_statement: STRUCT NAME newline struct_variable ENDSTRUCT
                | struct_statement STRUCT NAME newline struct_variable
ENDSTRUCT
                | TYPEDEF STRUCT USERDATATYPE newline struct_variable
USERDATATYPE ENDSTRUCT
                | struct_statement TYPEDEF STRUCT USERDATATYPE newline
struct_variable USERDATATYPE ENDSTRUCT
        ;

struct_variable: VARS datatype NAME ';' newline { if(variableSearch(variables,
$3) == 0){ variableListAppend(&variables, $3); } }
                | VARS datatype NAME ARRAY ';' newline {
if(variableSearch(variables, $3) == 0) { variableListAppend(&variables, $3); }
}
                | struct_variable VARS datatype NAME ';' newline {
if(variableSearch(variables, $4) == 0) { variableListAppend(&variables, $4); }
}
                | struct_variable VARS datatype NAME ARRAY ';' newline {
if(variableSearch(variables, $4) == 0) { variableListAppend(&variables, $4); }
}
        ;

function: function_declaration commands function_end newline /*{
printf("Function Statement found\n"); }*/
        | function_declaration variable_declaration commands function_end
newline /*{ printf("Function Statement found\n"); }*/
        | function function_declaration commands newline function_end newline
/*{ printf("Function Statement found\n"); }*/
        | function function_declaration variable_declaration commands
function_end newline /*{ printf("Function Statement found\n"); }*/
        ;

function_declaration: FUNCTION function_name newline /*{ printf("Function
Declared\n"); }*/
        ;

function_name: NAME '(' function_arguments ')' { if(functionSearch(functions,
$1) == 0) { functionListAppend(&functions, $1); } }
        ;

function_statement: NAME '(' variable ')' { if(functionSearch(functions, $1)
== 0) { printf("\nERROR: Function %s is NOT declared, in line
%d\n", $1, yylineno); fprintf(diagnostics, "ERROR: Function %s is NOT declared,
in line %d\n", $1, yylineno); YYABORT; } }
```

```
        ;

function_arguments: NAME { if(variableSearch(variables, $1) == 0) {
variableListAppend(&variables, $1); } }
        | function_arguments ',' function_arguments
        ;

function_end: RETURN NUM END_FUNCTION
        | RETURN CHARLITERAL END_FUNCTION
        | RETURN variable END_FUNCTION
        ;

main_statement: STARTMAIN commands ENDMAIN /*{ printf("MAIN Statement
found\n"); }*/
        | STARTMAIN variable_declaration commands ENDMAIN /*{
printf("MAIN Statement found\n"); }*/
        ;

command: assignment
        | print_statement
        | loop_statement
        | break_command
        | control_statement
        ;

commands: command newline
        | commands command newline
        ;

assignment: variable '=' expression ';'
        | assignment variable '=' expression ';'
        ;

loop_statement: while_statement
        | for_statement
        ;

break_command: BREAK ';' /*{ printf("Break command found\n"); }*/
        ;

control_statement: if_statement
        | switch_statement
        ;

while_statement: WHILE '(' condition ')' newline commands ENDWHILE /*{
printf("While Loop statement found\n"); }*/
```

```

;

condition: expression COMP_OPERATOR expression
          | expression AND expression
          | expression OR expression
          | '(' condition ')' COMP_OPERATOR '(' condition ')'
          | '(' condition ')' AND '(' condition ')'
          | '(' condition ')' OR '(' condition ')'
;

for_statement: FOR NAME ASSIGN_OPERATOR NUM TO NUM STEP NUM newline commands
ENDFOR { if(variableSearch(variables, $<str>2) == 0) { printf("\nERROR:
Variable %s is NOT declared, in line %d\n", $<str>2, yylineno);
fprintf(diagnostics, "ERROR: Variable %s is NOT declared, in line
%d\n", $<str>2, yylineno); YYABORT; } }
;

if_statement: IF '(' condition ')' THEN newline commands ENDIF /*{ printf("If
statement found\n"); }*/
          | IF '(' condition ')' THEN newline commands else_if_statement
ENDIF /*{ printf("If statement found\n"); }*/
          | IF '(' condition ')' THEN newline commands else_statement ENDIF
/*{ printf("If statement found\n"); }*/
          | IF '(' condition ')' THEN newline commands else_if_statement
else_statement ENDIF /*{ printf("If statement found\n"); }*/
;

else_if_statement: ELSEIF '(' condition ')' newline commands
                  | else_if_statement ELSEIF '(' condition ')' newline commands
;

else_statement: ELSE newline commands
;

switch_statement: SWITCH '(' expression ')' newline case ENDSWITCH /*{
printf("Switch statement found\n"); }*/
                | SWITCH '(' expression ')' newline case default ENDSWITCH /*{
printf("Switch statement found\n"); }*/
;

case: CASE '(' expression ')' ':' newline commands
     | case CASE '(' expression ')' ':' newline commands
;

default: DEFAULT ':' newline commands
;
```



```
print_statement: PRINT '(' STRLITERAL ')' ';' /*{ printf("Print Statement
found\n"); }*/
                | PRINT '(' STRLITERAL ',' variable ')' ';' /*{ printf("Print
Statement found\n"); }*/
                ;

expression: literal
          | variable
          | function_statement
          | expression '+' expression
          | expression '-' expression
          | expression '^' expression
          | expression '*' expression
          | expression '/' expression
          | '(' expression ')'
          ;

variable: NAME { if(variableSearch(variables, $<str>1) == 0) {
printf("\nERROR: Variable %s is NOT declared, in line %d\n", $<str>1,yylineno);
fprintf(diagnostics,"ERROR: Variable %s is NOT declared, in line
%d\n", $<str>1,yylineno); YYABORT; } }
          | NAME ARRAY { if(variableSearch(variables, $<str>1) == 0) {
printf("\nERROR: Variable %s is NOT declared, in line %d\n", $<str>1,yylineno);
fprintf(diagnostics,"ERROR: Variable %s is NOT declared, in line
%d\n", $<str>1,yylineno); YYABORT; } }
          | variable ',' variable
          ;

datatype: DATATYPE
        | USERDATATYPE
        ;

literal: NUM
        | STRLITERAL
        | CHARLITERAL
        ;

newline: NEWLINE /*{ line++; }*/
        ;

%%

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

```
fprintf(diagnostics, "ERROR: %s in line %d\n", s, yylineno);
}

int main (int argc, char **argv)
{
    ++argv; --argc;
    if ( argc > 0 )
        yyin = fopen( argv[0], "r" );
    else
        yyin = stdin;
    yyout = fopen ( "output.c", "w" );

    printf("C-like parser, implemented by Christos-Panagiotis Mpalatsouras,
Student ID = 1054335\n");
    printf("Program source code from parser input is following: \n\n");

    diagnostics = fopen("diagnostics.txt", "w");
    fprintf(diagnostics, "***** START of Diagnostic Messages *****\n\n");

    yyparse();

    printf("Program Name: %s\n", progr);
    fprintf(diagnostics, "Program: %s\n\n", progr);
    printf("Lines of code parsed: %i\n", line);
    fprintf(diagnostics, "Lines of code parsed: %i\n", line);
    fprintf(diagnostics, "\n***** END of Diagnostic Messages *****\n");
    fclose(diagnostics);

    printf("\nVariable List Items: \n");
    printVariableList(variables);

    printf("\nFunction List Items: \n");
    printFunctionList(functions);

    free(variables);
    free(functions);

    return 0;
}
```

4.4.3 ΠΑΡΑΔΕΙΓΜΑ ΛΕΙΤΟΥΡΓΙΑΣ (TEST CASE):

Στο παράδειγμα λειτουργίας του ερωτήματος 2, προστίθενται οι επιπλέον δηλώσεις για όλες τις μεταβλητές που χρησιμοποιούνται στο πρόγραμμα, ώστε να μην εμφανίζεται σφάλμα δήλωσης μεταβλητών.

ΣΥΝΟΛΙΚΟΣ ΚΩΔΙΚΑΣ ΠΑΡΑΔΕΙΓΜΑΤΟΣ ΣΤΗΝ ΨΕΥΔΟΓΛΩΣΣΑ:

```
PROGRAM Sorting_Algorithms %This is a test program for lexer and parser development
purposes
TYPEDEF STRUCT Node
    VARS INTEGER value;
    VARS Node prev;
    VARS Node next;
Node ENDSTRUCT FUNCTION BubbleSort(K, m)
    VARS INTEGER i,j,head,head2,temp;
    VARS INTEGER K_i,K_i_plus_1,K_m;
    i = 0;
    j = 0;
    head = m;
    WHILE (head > 1)
        j = 0;
        head2 = head - 1;
        PRINT("Sorting elements 0 --> %d, ", head2);
        FOR i:=0 TO 20 STEP 1
            IF (K_i > K_i_plus_1) THEN
                temp = K_i_plus_1;
                K_i_plus_1 = K_i;
                K_i = temp;
                j = j + 1;
            ENDIF
        ENDFOR
        PRINT("Element swaps in the current table scan: %d\n", j);
        IF (j == 0) THEN
            PRINT("No swap during last table scan. The table sorting completed
successfully\n");
            BREAK;
        ENDIF
        head = head - 1;
    ENDWHILE
RETURN K_m END_FUNCTION
FUNCTION InsertionSort(K, m)
    VARS INTEGER i,j,k,K_i,K_i_plus_1,K_j,K_m;
    i = 0;
    j = 0;
    k = 0;
    FOR j:=2 TO 20 STEP 1
        PRINT("Sorting elements 0 --> %d\n", j);
        k = K_j;
        i = j - 1;
```

```
        WHILE ((i > 0) AND (k < K_i))
            K_i_plus_1 = K_i;
            i = i - 1;
        ENDWHILE
        K_i_plus_1 = k;
    ENDFOR
RETURN K_m END_FUNCTION
STARTMAIN VARS INTEGER n,i,c,A[20];
    VARS INTEGER values[20];
    VARS INTEGER option;
    VARS INTEGER A_n,A_i;
    n = 20;
    A[20] = values;
    PRINT("Initial Table: \n");
    FOR i:=0 TO 20 STEP 1
        PRINT("%d ", A_i);
    ENDFOR
    PRINT("\nOptions: \n");
    PRINT("1. Bubble-Sort:\n");
    PRINT("2. Insertion-Sort:\n");
    PRINT("3. Selection-Sort:\n");
    PRINT("Select Algorithm >> "); %scanf an option from keyboard
    SWITCH (option)
    CASE (1):
        A_n = BubbleSort(A,n);
        PRINT("Results: \n");
        FOR i:=0 TO 20 STEP 1
            PRINT("%d ", A_i);
        ENDFOR
        PRINT("\n");
    CASE (2):
        A_n = InsertionSort(A,n);
        PRINT("Results: \n");
        FOR i:=0 TO 20 STEP 1
            PRINT("%d ", A_i);
        ENDFOR
        PRINT("\n");
    DEFAULT:
        PRINT("No option selected\n");
    ENDSWITCH
ENDMAIN
```

ΠΑΡΑΔΕΙΓΜΑ ΕΠΙΤΥΧΟΥΣ ΕΚΤΕΛΕΣΗΣ:

Δίνοντας ως είσοδο τον παραπάνω δοκιμαστικό κώδικα στον αναλυτή που υλοποιήθηκε στα πλαίσια της εργασίας, προκύπτει το παρακάτω αποτέλεσμα στα στιγμιότυπα οθόνης που ακολουθούν:

```
takis@takis-VirtualBox: ~/parser
takis@takis-VirtualBox:~/parser$ bison -y -d c-like_parser.y
takis@takis-VirtualBox:~/parser$ flex c-like_lexer.l
takis@takis-VirtualBox:~/parser$ gcc y.tab.c lex.yy.c -lfl
y.tab.c: In function 'yyparse':
y.tab.c:1596:16: warning: implicit declaration of function 'yylex' [-Wimplicit-function-declaration]
1596 |         yychar = yylex ();
      |         ^~~~~~
y.tab.c:1858:18: warning: passing argument 1 of 'yyerror' discards 'const' qualifier from pointer target type [-Wdiscarded-qualifiers]
1858 |         yyerror (yymsgp);
      |         ^~~~~~
c-like_parser.y:9:14: note: expected 'char *' but argument is of type 'const char *'
9 | void yyerror(char *);
  | ~~~~~~
takis@takis-VirtualBox:~/parser$ ./a.out prog.c
c-like parser, implemented by Christos-Panagiotis Mpalatsouras, Student ID = 1054335
Program source code from parser input is following:

PROGRAM Sorting_Algorithms %This is a test program for lexer and parser development purposes
TYPEDEF STRUCT Node
  VARS INTEGER value;
  VARS Node prev;
  VARS Node next;
Node ENDSTRUCT FUNCTION BubbleSort(K, m)
  VARS INTEGER i, j, head, head2, temp;
  VARS INTEGER K_i, K_i_plus_1, K_m;
  i = 0;
  j = 0;
  head = m;
  WHILE (head > 1)
    j = 0;
    head2 = head - 1;
    PRINT("Sorting elements 0 --> %d, ", head2);
    FOR i:=0 TO 20 STEP 1
      IF (K_i > K_i_plus_1) THEN
        temp = K_i_plus_1;
        K_i_plus_1 = K_i;
        K_i = temp;
        j = j + 1;
      ENDIF
    ENDFOR
    PRINT("Element swaps in the current table scan: %d\n", j);
    IF (j == 0) THEN
      PRINT("No swap during last table scan. The table sorting completed successfully\n");
      BREAK;
    ENDIF
    head = head - 1;
  ENDWHILE
RETURN K_m END_FUNCTION
FUNCTION InsertionSort(K, m)
  VARS INTEGER i, j, k, K_i, K_i_plus_1, K_j, K_m;
  i = 0;
  j = 0;
  k = 0;
  FOR j:=2 TO 20 STEP 1
    PRINT("Sorting elements 0 --> %d\n", j);
    k = K_j;
    i = j - 1;
    WHILE ((i > 0) AND (k < K_i))
      K_i_plus_1 = K_i;
      i = i - 1;
    ENDWHILE
    K_i_plus_1 = k;
  ENDFOR
RETURN K_m END_FUNCTION
STARTMAIN VARS INTEGER n, i, c, A[20];
  VARS INTEGER values[20];
  VARS INTEGER option;
  VARS INTEGER A_n, A_i;
  n = 20;
  A[20] = values;
  PRINT("Initial Table: \n");
  FOR i:=0 TO 20 STEP 1
    PRINT("%d ", A_i);
  ENDFOR
  PRINT("\nOptions: \n");
  PRINT("1. Bubble-Sort:\n");
  PRINT("2. Insertion-Sort:\n");
  PRINT("3. Selection-Sort:\n");
  PRINT("Select Algorithm >> "); %scanf an option from keyboard
  SWITCH (option)
  CASE (1):
    A_n = BubbleSort(A, n);
    PRINT("Results: \n");
    FOR i:=0 TO 20 STEP 1
      PRINT("%d ", A_i);
    ENDFOR
    PRINT("\n");
  CASE (2):
    A_n = InsertionSort(A, n);
    PRINT("Results: \n");
    FOR i:=0 TO 20 STEP 1
      PRINT("%d ", A_i);
    ENDFOR
    PRINT("\n");
  DEFAULT:
    PRINT("No option selected\n");
  ENDSWITCH
ENDMAIN
```

```
takis@takis-VirtualBox: ~/parser
  VARS INTEGER i, j, k, K_i, K_i_plus_1, K_j, K_m;
  i = 0;
  j = 0;
  k = 0;
  FOR j:=2 TO 20 STEP 1
    PRINT("Sorting elements 0 --> %d\n", j);
    k = K_j;
    i = j - 1;
    WHILE ((i > 0) AND (k < K_i))
      K_i_plus_1 = K_i;
      i = i - 1;
    ENDWHILE
    K_i_plus_1 = k;
  ENDFOR
RETURN K_m END_FUNCTION
STARTMAIN VARS INTEGER n, i, c, A[20];
  VARS INTEGER values[20];
  VARS INTEGER option;
  VARS INTEGER A_n, A_i;
  n = 20;
  A[20] = values;
  PRINT("Initial Table: \n");
  FOR i:=0 TO 20 STEP 1
    PRINT("%d ", A_i);
  ENDFOR
  PRINT("\nOptions: \n");
  PRINT("1. Bubble-Sort:\n");
  PRINT("2. Insertion-Sort:\n");
  PRINT("3. Selection-Sort:\n");
  PRINT("Select Algorithm >> "); %scanf an option from keyboard
  SWITCH (option)
  CASE (1):
    A_n = BubbleSort(A, n);
    PRINT("Results: \n");
    FOR i:=0 TO 20 STEP 1
      PRINT("%d ", A_i);
    ENDFOR
    PRINT("\n");
  CASE (2):
    A_n = InsertionSort(A, n);
    PRINT("Results: \n");
    FOR i:=0 TO 20 STEP 1
      PRINT("%d ", A_i);
    ENDFOR
    PRINT("\n");
  DEFAULT:
    PRINT("No option selected\n");
  ENDSWITCH
ENDMAIN
```

```
takis@takis-VirtualBox: ~/parser

PRINT("Results: \n");
FOR i:=0 TO 20 STEP 1
    PRINT("kd ", A_i);
ENDFOR
PRINT("\n");
CASE (2):
    A.n = InsertionSort(A,n);
    PRINT("Results: \n");
    FOR i:=0 TO 20 STEP 1
        PRINT("kd ", A_i);
    ENDFOR
    PRINT("\n");
DEFAULT:
    PRINT("No option selected\n");
ENDSWITCH
ENDMAIN

Code Parsed successfully!
Program Name: Sorting_Algorithms
Lines of code parsed: 81

Variable List Items:
Variable Name: next
Variable Name: prev
Variable Name: value
Variable Name: K
Variable Name: m
Variable Name: i
Variable Name: j
Variable Name: head
Variable Name: head2
Variable Name: temp
Variable Name: K_i
Variable Name: K_i_plus_1
Variable Name: K_m
Variable Name: k
Variable Name: K_j
Variable Name: n
Variable Name: c
Variable Name: A
Variable Name: values
Variable Name: option
Variable Name: A.n
Variable Name: A_i

Function List Items:
Function Name: BubbleSort
Function Name: InsertionSort
takis@takis-VirtualBox: ~/parser$
```

Επειδή το παραπάνω αποτέλεσμα της εκτέλεσης του αναλυτή δεν χώραγε σε μια οθόνη, παραπάνω παρουσιάζονται τρία στιγμιότυπα οθόνης όπου το επόμενο αποτελεί συνέχεια του προηγούμενου.

Ο κώδικας της εισόδου, έχει επιστραφεί στην έξοδο στο παραπάνω τερματικό και στο αρχείο output.c:

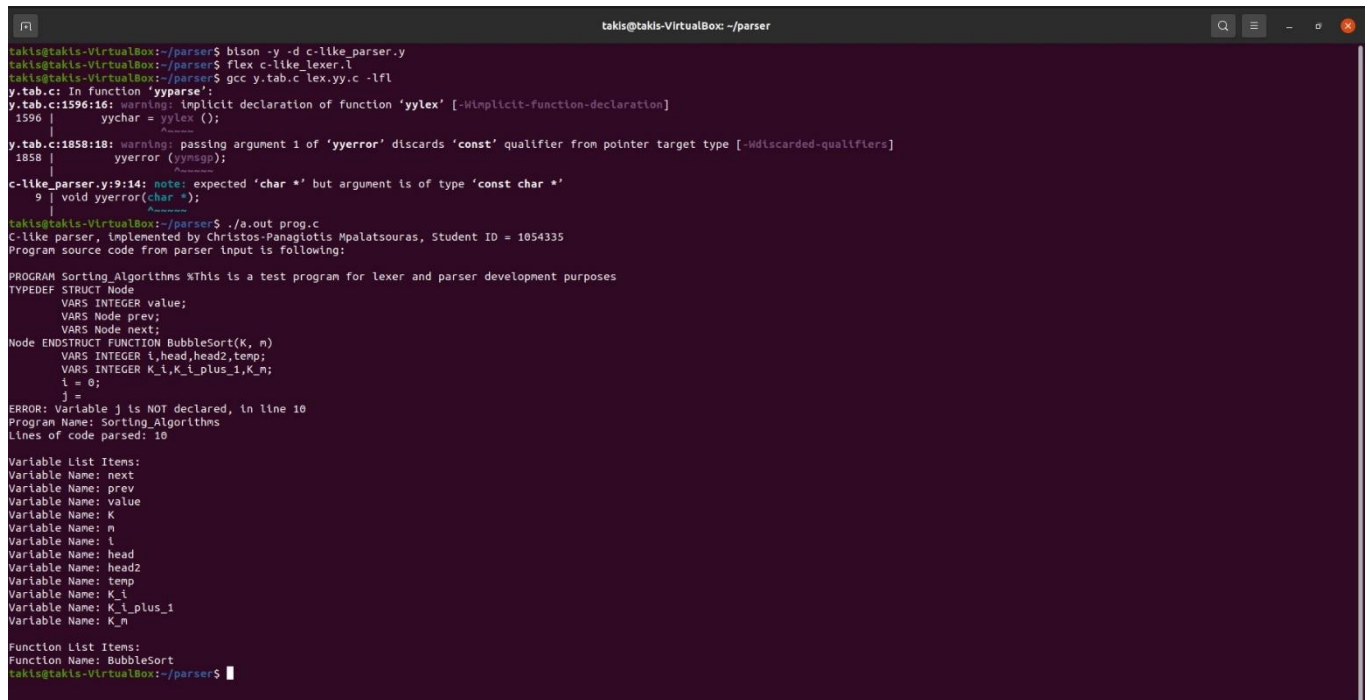
```
Ανοίγμα  output.c  Αποθήκευση
1 PROGRAM Sorting_Algorithms %This is a test program for lexer and parser development purposes
2 TYPEDEF STRUCT Node
3     VARS INTEGER value;
4     VARS Node prev;
5     VARS Node next;
6 Node ENDSTRUCT FUNCTION BubbleSort(K, m)
7     VARS INTEGER i,j,head,head2,temp;
8     VARS INTEGER K_i,K_i_plus_1,K_m;
9     i = 0;
10    j = 0;
11    head = m;
12    WHILE (head > 1)
13        j = 0;
14        head2 = head - 1;
15        PRINT("Sorting elements 0 --> %d, ", head2);
16        FOR i:=0 TO 20 STEP 1
17            IF (K_i > K_i_plus_1) THEN
18                temp = K_i_plus_1;
19                K_i_plus_1 = K_i;
20                K_i = temp;
21                j = j + 1;
22            ENDIF
23        ENDFOR
24        PRINT("Element swaps in the current table scan: %d\n", j);
25        IF (j == 0) THEN
26            PRINT("No swap during last table scan. The table sorting completed successfully\n");
27            BREAK;
28        ENDIF
29        head = head - 1;
30    ENDWHILE
31 RETURN K_m END_FUNCTION
32 FUNCTION InsertionSort(K, m)
33     VARS INTEGER i,j,k,K_i,K_i_plus_1,K_j,K_m;
34     i = 0;
35     j = 0;
36     K = 0;
37     FOR j:=2 TO 20 STEP 1
38         PRINT("Sorting elements 0 --> %d\n", j);
39         k = K_j;
40         i = j - 1;
41         WHILE ((i > 0) AND (k < K_i))
42             K_i_plus_1 = K_i;
43             i = i - 1;
44         ENDWHILE
45         K_i_plus_1 = k;
46     ENDFOR
47 RETURN K_m END_FUNCTION
48 STARTMAIN VARS INTEGER n,i,c,A[20];
49 MAKE_INTEGER_ARRAY c=0;
50
```

Ακολουθεί στην επόμενη σελίδα το παράδειγμα ανεπιτυχούς εκτέλεσης.

ΠΑΡΑΔΕΙΓΜΑ ΑΝΕΠΙΤΥΧΟΥΣ ΕΚΤΕΛΕΣΗΣ:

Έστω ότι στη γραμμή 10 δεν έχει δηλωθεί η μεταβλητή «j».

Εμφανίζεται το αντίστοιχο μήνυμα σφάλματος στο τερματικό και η ανάλυση σταματάει, όπως φαίνεται στο παρακάτω στιγμιότυπο οθόνης:



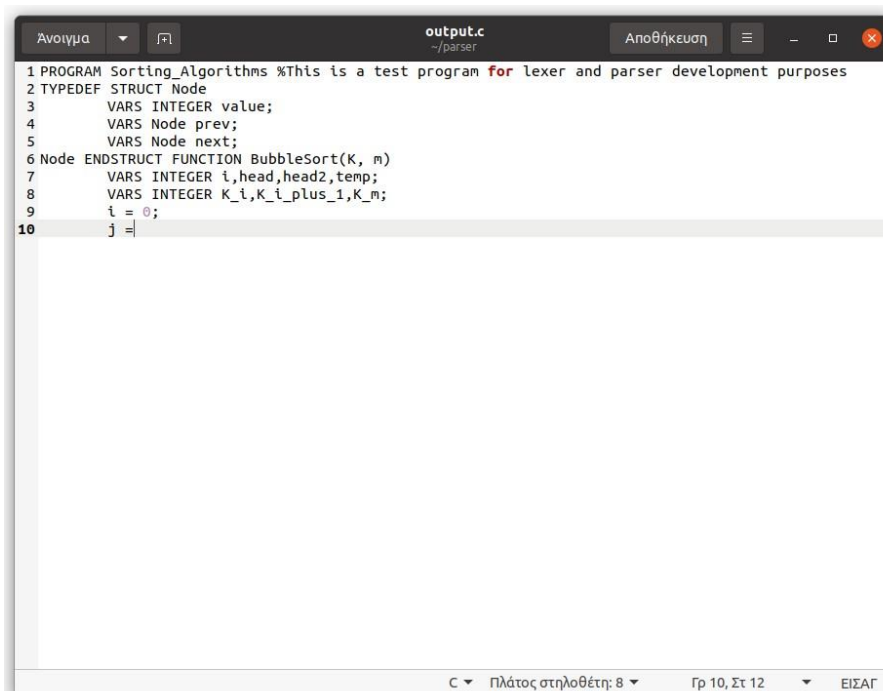
```
takis@takis-VirtualBox: ~/parser
takis@takis-VirtualBox:~/parser$ bison -y -d c-like.parser.y
takis@takis-VirtualBox:~/parser$ flex c-like.lexer.l
takis@takis-VirtualBox:~/parser$ gcc y.tab.c lex.yy.c -lfl
y.tab.c: In function 'yyparse':
y.tab.c:1596:16: warning: implicit declaration of function 'yylex' [-Wimplicit-function-declaration]
1596 |         yychar = yylex ();
      |         ^~~~~~
y.tab.c:1858:18: warning: passing argument 1 of 'yyerror' discards 'const' qualifier from pointer target type [-Wdiscarded-qualifiers]
1858 |         yyerror (yynsgp);
      |         ^~~~~~
c-like.parser.y:9:14: note: expected 'char *' but argument is of type 'const char *'
9 | void yyerror(char *);
  |               ^
takis@takis-VirtualBox:~/parser$ ./a.out prog.c
C-like parser, implemented by Christos-Panagiotis Mpatsouras, Student ID = 1054335
Program source code from parser input is following:

PROGRAM Sorting_Algorithms %This is a test program for lexer and parser development purposes
TYPEDEF STRUCT Node
  VARS INTEGER value;
  VARS Node prev;
  VARS Node next;
Node ENDSTRUCT FUNCTION BubbleSort(K, m)
  VARS INTEGER i, head, head2, temp;
  VARS INTEGER K_i, K_i_plus_1, K_m;
  i = 0;
  j =
ERROR: Variable j is NOT declared, in line 10
Program Name: Sorting_Algorithms
Lines of code parsed: 10

Variable List Items:
Variable Name: next
Variable Name: prev
Variable Name: value
Variable Name: K
Variable Name: m
Variable Name: i
Variable Name: head
Variable Name: head2
Variable Name: temp
Variable Name: K_i
Variable Name: K_i_plus_1
Variable Name: K_m

Function List Items:
Function Name: BubbleSort
takis@takis-VirtualBox:~/parser$
```

Ο κώδικας της εισόδου δεν έχει επιστραφεί ολόκληρος στην έξοδο, στο παραπάνω τερματικό και στο αρχείο output.c, καθώς έχει σταματήσει στη γραμμή που εμφανίζεται το σφάλμα.



```
output.c
~/parser
1 PROGRAM Sorting_Algorithms %This is a test program for lexer and parser development purposes
2 TYPEDEF STRUCT Node
3   VARS INTEGER value;
4   VARS Node prev;
5   VARS Node next;
6 Node ENDSTRUCT FUNCTION BubbleSort(K, m)
7   VARS INTEGER i, head, head2, temp;
8   VARS INTEGER K_i, K_i_plus_1, K_m;
9   i = 0;
10  j =
```

4.5 ΕΡΩΤΗΜΑ 4 – ΣΧΟΛΙΑ ΠΟΛΛΑΠΛΩΝ ΓΡΑΜΜΩΝ

Σε αυτή την ενότητα, τροποποιείται ο λεκτικός αναλυτής ώστε να υποστηρίζει σχόλια πολλαπλών γραμμών. Τα σχόλια πολλαπλών γραμμών έχουν τη μορφή:

```
/* έναρξη σχολίων
```

```
...
```

```
Οτιδήποτε
```

```
...
```

```
Τέλος σχολίων */
```

```
ή
```

```
/* σχόλια */
```

Για να υλοποιηθεί αυτή η δυνατότητα, δημιουργείται μια νέα κατάσταση στον flex.

Με τη χρήση των καταστάσεων, μπορεί να ενεργοποιηθεί ένας κανόνας υπό συνθήκη, δηλαδή μόνο όταν ο flex είναι σε αυτή την κατάσταση μπορεί να ενεργοποιήσει τον κανόνα που ανήκει στην κατάσταση. Αρχικά ο flex βρίσκεται στην κατάσταση INITIAL και μπορεί να μεταβεί σε κάποια κατάσταση με την εντολή «BEGIN(NEW_STATE);».

Για να μεταβεί ο flex σε κάποια κατάσταση, χρειάζεται στο action ενός κανόνα που ικανοποιεί τη συνθήκη για τη μετάβαση σε διαφορετική κατάσταση, να βρίσκεται η εντολή BEGIN. Ο Flex επιστρέφει ξανά στην αρχική κατάσταση με την εντολή «BEGIN(INITIAL);».

Συνοπτικά, στην περίπτωση αυτού του ερωτήματος, δημιουργείται μια νέα κατάσταση, η οποία ενεργοποιείται μόλις εντοπιστεί η ακολουθία “/*” στην είσοδο και μετά εκτελούνται οι κανόνες που ανήκουν σε αυτή την κατάσταση μέχρι να ενεργοποιηθεί η συνθήκη εξόδου. Όταν ενεργοποιηθεί ο κανόνας που ικανοποιεί την ακολουθία εισόδου “*/”, ο οποίος κανόνας αποτελεί τη συνθήκη εξόδου από τη νέα κατάσταση, στο action αυτού του κανόνα υπάρχει η εντολή για την επιστροφή του flex στην κατάσταση INITIAL. Τέλος, ο flex επιστρέφει στην αρχική κατάσταση και συνεχίζει τη λεκτική ανάλυση της εισόδου.

ΣΧΕΔΙΑΣΤΙΚΗ ΠΑΡΑΔΟΧΗ:

Μετά το τέλος ενός σχολίου πολλαπλών γραμμών, η συνέχεια του κώδικα της ψευδογλώσσας μπορεί να βρίσκεται είτε στη γραμμή που παρατηρείται το τέλος του σχολίου «*/» είτε στην επόμενη γραμμή, έπειτα από αλλαγή γραμμής.

4.5.1 ΤΡΟΠΟΠΟΙΗΣΗ ΤΟΥ ΛΕΞΙΚΟΥ ΑΝΑΛΥΤΗ (LEXER)

Για τους σκοπούς της υλοποίησης αυτού του ερωτήματος, γίνονται οι παρακάτω τροποποιήσεις στον κώδικα εισόδου του Flex για τον αναλυτή της εργασίας:

Αρχικά ορίζεται μια νέα κατάσταση με όνομα «MULTILINECOMM».

Στη συνέχεια, δημιουργείται ο κανόνας που αναγνωρίζει την ακολουθία χαρακτήρων «/*». Στο action αυτού του κανόνα βρίσκεται η εντολή «BEGIN(MULTILINECOMM);», η οποία ενεργοποιεί την κατάσταση «MULTILINECOMM».

Οι κανόνες που ανήκουν στη νέα κατάσταση είναι οι εξής:

Ο πρώτος με τη σειρά στον κώδικα κανόνας αναγνωρίζει και αγνοεί όλους τους χαρακτήρες εκτός από τον χαρακτήρα «*». Ο δεύτερος κανόνας αναγνωρίζει και αγνοεί οποιοδήποτε «*» το οποίο δεν ακολουθείται από το χαρακτήρα «/». Ο τρίτος κανόνας αναγνωρίζει και αγνοεί το χαρακτήρα αλλαγής γραμμής.

Ο τελευταίος κανόνας αποτελεί και τη συνθήκη εξόδου, αναγνωρίζει είτε την ακολουθία «*/» είτε την ακολουθία «*/» ακολουθούμενη από αλλαγή γραμμής, ώστε ο κώδικας της ψευδογλώσσας να μπορεί να συνεχίζει και στην επόμενη γραμμή από την τελευταία γραμμή του σχολίου πολλαπλών γραμμών. Στο action αυτού του κανόνα βρίσκεται η εντολή «BEGIN(INITIAL)» ώστε ο flex να επιστρέψει στην αρχική κατάσταση και να συνεχίσει η κανονική λεκτική ανάλυση της εισόδου.

ΕΝΗΜΕΡΩΜΕΝΟΣ ΚΩΔΙΚΑΣ ΕΙΣΟΔΟΥ ΣΤΟ FLEX (ΑΡΧΕΙΟ ΕΙΣΟΔΟΥ c-like_lexer.l):

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

%option yylineno

underscore "_"
digit      [0-9]
letter     [a-zA-Z]
chartype   "CHAR"
inttype    "INTEGER"
userdatatype "Node"
comparative ">" | "<" | "==" | "!="
char       .
charsequence {char}*
strliteral \"{charsequence}\"
```

```

charliteral      "{char}"
linecomment      "%"{charsequence}
datatype         {chartype}|{inttype}
basiccharacter   {underscore}|{letter}|{digit}
number           {digit}+|{digit}+".{digit}+
basicstring      {basiccharacter}*
name             {letter}{basicstring}|{underscore}{basicstring}
array            "["{number}"]"

%x MULTILINECOMM
%%

[ \t]+ { yylval.str = strdup(yytext); ECHO; printf("%s",yytext);}
"\n" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
NEWLINE; }
";" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return ';' ; }
":" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return ':' ; }
"," { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return ',' ; }
"(" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return '(' ; }
")" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return ')' ; }
"+" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return '+' ; }
"-" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return '-' ; }
"^" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return '^' ; }
"*" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return '*' ; }
"/" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return '/' ; }
"=" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return '=' ; }
"AND" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return AND; }
"OR" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return OR; }
":=" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
ASSIGN_OPERATOR; }
"PROGRAM" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
PROGRAM; }
"VARS" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
VARS; }
"TYPEDEF" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
TYPEDEF; }
"STRUCT" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
STRUCT; }
"ENDSTRUCT" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext);
return ENDSSTRUCT; }
"FUNCTION" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
FUNCTION; }
"END_FUNCTION" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext);
return END_FUNCTION; }

```

```
"RETURN" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return  
RETURN; }  
"WHILE" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return  
WHILE; }  
"ENDWHILE" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return  
ENDWHILE; }  
"FOR" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return FOR; }  
"TO" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return TO; }  
"STEP" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return STEP;  
}  
"ENDFOR" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return  
ENDFOR; }  
"IF" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return IF; }  
"THEN" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return THEN;  
}  
"ELSEIF" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return  
ELSEIF; }  
"ELSE" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return ELSE;  
}  
"ENDIF" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return  
ENDIF; }  
"SWITCH" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return  
SWITCH; }  
"CASE" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return CASE;  
}  
"DEFAULT" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return  
DEFAULT; }  
"ENDSWITCH" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return  
ENDSWITCH; }  
"PRINT" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return  
PRINT; }  
"BREAK" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return  
BREAK; }  
"STARTMAIN" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return  
STARTMAIN; }  
"ENDMAIN" { yylval.str = strdup(yytext); ECHO; printf("%s\n",yytext); return  
ENDMAIN; }  
{strliteral} { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return  
STRLITERAL; }  
{charliteral} { yylval.str = strdup(yytext); ECHO; printf("%s",yytext);  
return CHARLITERAL; }  
{linecomment} { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); }  
{userdatatype} { yylval.str = strdup(yytext); ECHO; printf("%s",yytext);  
return USERDATATYPE; }  
{datatype} { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return  
DATATYPE; }
```

```
{name} { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return NAME;
}
{number} { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
NUM; }
{array} { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
ARRAY; }
{comparative} { yylval.str = strdup(yytext); ECHO; printf("%s",yytext);
return COMP_OPERATOR; }
"/*" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext);
BEGIN(MULTILINECOMM);}
<MULTILINECOMM>[^*\n]* { yylval.str = strdup(yytext); ECHO;
printf("%s",yytext); }
<MULTILINECOMM>"*"+[^\n]* { yylval.str = strdup(yytext); ECHO;
printf("%s",yytext); }
<MULTILINECOMM>"\n" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext);
}
<MULTILINECOMM>"*"+"/"|"*"+"/"+" \n" { yylval.str = strdup(yytext); ECHO;
printf("%s",yytext); BEGIN(INITIAL); }

%%
```

Στην επόμενη σελίδα βρίσκονται τα παραδείγματα λειτουργίας για το ερώτημα 4.

4.5.2 ΠΑΡΑΔΕΙΓΜΑ ΛΕΙΤΟΥΡΓΙΑΣ (TEST CASE):

Στο παράδειγμα λειτουργίας του ερωτήματος 3, προστίθενται οι επιπλέον γραμμές που περιέχουν σχόλια πολλαπλών γραμμών.

ΣΥΝΟΛΙΚΟΣ ΚΩΔΙΚΑΣ ΠΑΡΑΔΕΙΓΜΑΤΟΣ ΣΤΗΝ ΨΕΥΔΟΓΛΩΣΣΑ:

```
/* Created by Christos-Panagiotis Balatsouras, StudentID = 1054335
   CEID - University of Patras */
/*Program declaration*/
PROGRAM Sorting_Algorithms %This is a test program for lexer and parser development
purposes
/*Struct declaration*/
TYPEDEF STRUCT Node
    VARS INTEGER value;
    VARS Node prev;
    VARS Node next;
Node ENDSTRUCT /*Function declaration*/ FUNCTION BubbleSort(K, m)
    VARS INTEGER i,j,head,head2,temp;
    VARS INTEGER K_i,K_i_plus_1,K_m;
    i = 0;
    j = 0;
    head = m;
    WHILE (head > 1)
        j = 0;
        head2 = head - 1;
        PRINT("Sorting elements 0 --> %d, ", head2);
        FOR i:=0 TO 20 STEP 1
            IF (K_i > K_i_plus_1) THEN
                temp = K_i_plus_1;
                K_i_plus_1 = K_i;
                K_i = temp;
                j = j + 1;
            ENDIF
        ENDFOR
        PRINT("Element swaps in the current table scan: %d\n", j);
        IF (j == 0) THEN
            PRINT("No swap during last table scan. The table sorting completed
successfully\n");
            BREAK;
        ENDIF
        head = head - 1;
    ENDWHILE
RETURN K_m END_FUNCTION
FUNCTION InsertionSort(K, m)
    VARS INTEGER i,j,k,K_i,K_i_plus_1,K_j,K_m;
    i = 0;
    j = 0;
    k = 0;
    FOR j:=2 TO 20 STEP 1
```

```
        PRINT("Sorting elements 0 --> %d\n", j);
        k = K_j;
        i = j - 1;
        WHILE ((i > 0) AND (k < K_i))
            K_i_plus_1 = K_i;
            i = i - 1;
        ENDWHILE
        K_i_plus_1 = k;
    ENDFOR
RETURN K_m END_FUNCTION
/*Main part of program*/
STARTMAIN VARS INTEGER n,i,c,A[20];
    VARS INTEGER values[20];
    VARS INTEGER option;
    VARS INTEGER A_n,A_i;
    n = 20;
    A[20] = values;
    PRINT("Initial Table: \n");
    FOR i:=0 TO 20 STEP 1
        PRINT("%d ", A_i);
    ENDFOR
    PRINT("\nOptions: \n");
    PRINT("1. Bubble-Sort:\n");
    PRINT("2. Insertion-Sort:\n");
    PRINT("3. Selection-Sort:\n");
    PRINT("Select Algorithm >> "); %scanf an option from keyboard
    SWITCH (option)
    CASE (1):
        A_n = BubbleSort(A,n);
        PRINT("Results: \n");
        FOR i:=0 TO 20 STEP 1
            PRINT("%d ", A_i);
        ENDFOR
        PRINT("\n");
    CASE (2):
        A_n = InsertionSort(A,n);
        PRINT("Results: \n");
        FOR i:=0 TO 20 STEP 1
            PRINT("%d ", A_i);
        ENDFOR
        PRINT("\n");
    DEFAULT:
        PRINT("No option selected\n");
    ENDSWITCH
ENDMAIN
```

ΠΑΡΑΔΕΙΓΜΑ ΕΠΙΤΥΧΟΥΣ ΕΚΤΕΛΕΣΗΣ:

Δίνοντας ως είσοδο τον παραπάνω δοκιμαστικό κώδικα στον αναλυτή που υλοποιήθηκε στα πλαίσια της εργασίας, προκύπτει το παρακάτω αποτέλεσμα στα στιγμιότυπα οθόνης που ακολουθούν:

```
takis@takis-VirtualBox: ~/parser
takis@takis-VirtualBox:~/parser$ bison -y -d c-like_parser.y
takis@takis-VirtualBox:~/parser$ flex c-like_lexer.l
takis@takis-VirtualBox:~/parser$ gcc -o myparser y.tab.c lex.yy.c -lfl
y.tab.c: In function 'yyparse':
y.tab.c:1596:16: warning: implicit declaration of function 'yylex' [-Wimplicit-function-declaration]
1596 |         yychar = yylex ();
      |         ^~~~~~
y.tab.c:1852:18: warning: passing argument 1 of 'yyerror' discards 'const' qualifier from pointer target type [-Wdiscarded-qualifiers]
1852 |         yyerror (yynsgpp);
      |         ^~~~~~
c-like_parser.y:9:14: note: expected 'char *' but argument is of type 'const char *'
9 | void yyerror(char *);
  |         ^~~~~~
takis@takis-VirtualBox:~/parser$ ./myparser prog.c
c-like parser, implemented by Christos-Panagiotis Mpalatsouras, Student ID = 1054335
Program source code from parser input is following:

/* Created by Christos-Panagiotis Balatsouras, StudentID = 1054335
   CEID - University of Patras */
/*Program declaration*/
PROGRAM Sorting_Algorithms %This is a test program for lexer and parser development purposes
/*struct declaration*/
/*typedef STRUC Node
    VARS INTEGER value;
    VARS Node prev;
    VARS Node next;
Node ENDSTRUCT */FUNCTION BubbleSort(K, n)
    VARS INTEGER i,j,head,head2,temp;
    VARS INTEGER K_i,K_i_plus_1,K_m;
    i = 0;
    j = 0;
    head = m;
    WHILE (head > 1)
        j = 0;
        head2 = head - 1;
        PRINT("Sorting elements 0 --> %d, ", head2);
        FOR i:=0 TO 20 STEP 1
            IF (K_i > K_i_plus_1) THEN
                temp = K_i_plus_1;
                K_i_plus_1 = K_i;
                K_i = temp;
                j = j + 1;
            ENDIF
        ENDFOR
        PRINT("Element swaps in the current table scan: %d\n", j);
        IF (j == 0) THEN
            PRINT("No swap during last table scan. The table sorting completed successfully\n");
            BREAK;
        ENDIF
        head = head - 1;
    ENDWHILE
RETURN K_m END_FUNCTION
FUNCTION InsertionSort(K, n)
    VARS INTEGER i,j,k,K_i,K_i_plus_1,K_j,K_m;
    i = 0;
    j = 0;
    k = 0;
    FOR j:=2 TO 20 STEP 1
        PRINT("Sorting elements 0 --> %d\n", j);
        k = K_j;
        i = j - 1;
        WHILE ((i > 0) AND (k < K_i))
            K_i_plus_1 = K_i;
            i = i - 1;
        ENDWHILE
        K_i_plus_1 = k;
    ENDFOR
RETURN K_m END_FUNCTION
/*Main part of program*/
STARTMAIN VARS INTEGER n,i,c,A[20];
    VARS INTEGER values[20];
    VARS INTEGER option;
    VARS INTEGER A_n,A_i;
    n = 20;
    A[20] = values;
    PRINT("Initial Table: \n");
    FOR i:=0 TO 20 STEP 1
        PRINT("%d ", A_i);
    ENDFOR
    PRINT("\nOptions: \n");
    PRINT("1. Bubble-Sort:\n");
    PRINT("2. Insertion-Sort:\n");
    PRINT("3. Selection-Sort:\n");
    PRINT("Select Algorithm >> "); %scanf an option from keyboard
    SWITCH (option)
    CASE (1):
        A_n = BubbleSort(A,n);
        PRINT("Results: \n");
        FOR i:=0 TO 20 STEP 1
            PRINT("%d ", A_i);
        ENDFOR
        PRINT("\n");
    CASE (2):
        A_n = InsertionSort(A,n);
        PRINT("Results: \n");
        FOR i:=0 TO 20 STEP 1
            PRINT("%d ", A_i);
        ENDFOR
        PRINT("\n");
    END
```

```
takis@takis-VirtualBox: ~/parser
head = head - 1;
ENDWHILE
RETURN K_m END_FUNCTION
FUNCTION InsertionSort(K, n)
    VARS INTEGER i,j,k,K_i,K_i_plus_1,K_j,K_m;
    i = 0;
    j = 0;
    k = 0;
    FOR j:=2 TO 20 STEP 1
        PRINT("Sorting elements 0 --> %d\n", j);
        k = K_j;
        i = j - 1;
        WHILE ((i > 0) AND (k < K_i))
            K_i_plus_1 = K_i;
            i = i - 1;
        ENDWHILE
        K_i_plus_1 = k;
    ENDFOR
RETURN K_m END_FUNCTION
/*Main part of program*/
STARTMAIN VARS INTEGER n,i,c,A[20];
    VARS INTEGER values[20];
    VARS INTEGER option;
    VARS INTEGER A_n,A_i;
    n = 20;
    A[20] = values;
    PRINT("Initial Table: \n");
    FOR i:=0 TO 20 STEP 1
        PRINT("%d ", A_i);
    ENDFOR
    PRINT("\nOptions: \n");
    PRINT("1. Bubble-Sort:\n");
    PRINT("2. Insertion-Sort:\n");
    PRINT("3. Selection-Sort:\n");
    PRINT("Select Algorithm >> "); %scanf an option from keyboard
    SWITCH (option)
    CASE (1):
        A_n = BubbleSort(A,n);
        PRINT("Results: \n");
        FOR i:=0 TO 20 STEP 1
            PRINT("%d ", A_i);
        ENDFOR
        PRINT("\n");
    CASE (2):
        A_n = InsertionSort(A,n);
        PRINT("Results: \n");
        FOR i:=0 TO 20 STEP 1
            PRINT("%d ", A_i);
        ENDFOR
        PRINT("\n");
    END
```

```
takis@takis-VirtualBox: ~/parser

PRINT("Results: \n");
FOR i:=0 TO 20 STEP 1
    PRINT("x", A_i);
ENDFOR
PRINT("\n");
CASE (2):
    A.n = InsertionSort(A,n);
    PRINT("Results: \n");
    FOR i:=0 TO 20 STEP 1
        PRINT("x", A_i);
    ENDFOR
    PRINT("\n");
DEFAULT:
    PRINT("No option selected\n");
ENDSWITCH
ENDMAIN

Code Parsed successfully!
Program Name: Sorting_Algorithms
Lines of code scanned and parsed: 87

Variable List Items:
Variable Name: next
Variable Name: prev
Variable Name: value
Variable Name: K
Variable Name: n
Variable Name: i
Variable Name: j
Variable Name: head
Variable Name: head2
Variable Name: temp
Variable Name: K_i
Variable Name: K_i_plus_1
Variable Name: K_m
Variable Name: k
Variable Name: K_j
Variable Name: n
Variable Name: c
Variable Name: A
Variable Name: values
Variable Name: option
Variable Name: A_n
Variable Name: A_i

Function List Items:
Function Name: BubbleSort
Function Name: InsertionSort
takis@takis-VirtualBox: ~/parser$
```

Επειδή το παραπάνω αποτέλεσμα της εκτέλεσης του αναλυτή δεν χώραγε σε μια οθόνη, παραπάνω παρουσιάζονται τρία στιγμιότυπα οθόνης όπου το επόμενο αποτελεί συνέχεια του προηγούμενου.

Ο κώδικας της εισόδου, έχει επιστραφεί στην έξοδο στο παραπάνω τερματικό και στο αρχείο output.c:

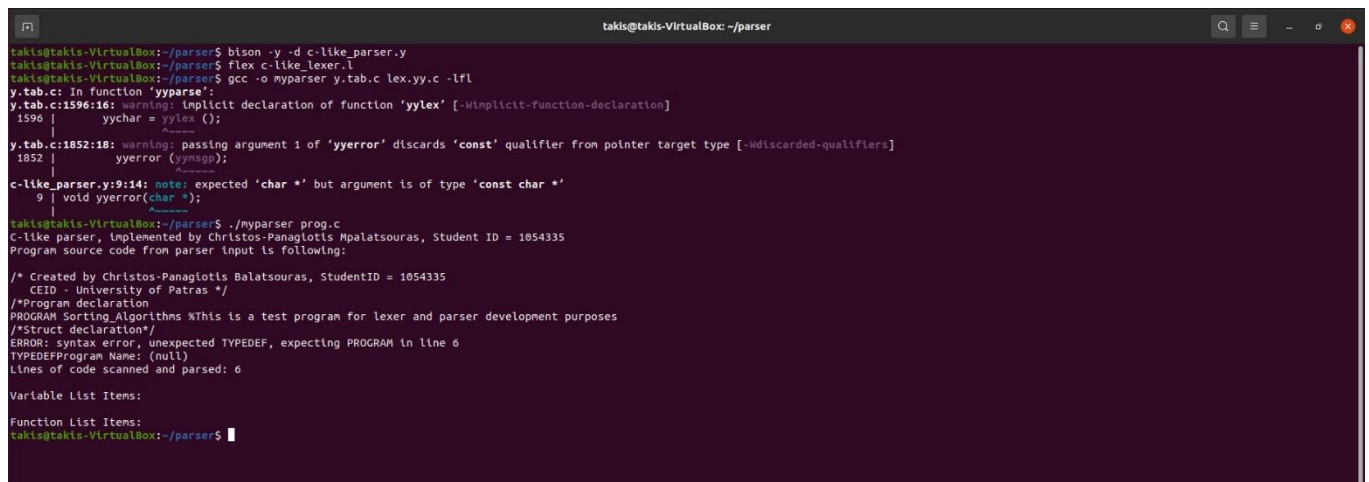
```
Ανοίγγω output.c
1 /* Created by Christos-Panagiotis Balatsouras, StudentID = 1054335
2  * CEID - University of Patras */
3 /*Program declaration*/
4 PROGRAM Sorting_Algorithms %This is a test program for lexer and parser development purposes
5 /*struct declaration*/
6 TYPEDEF STRUCT Node
7     VARS INTEGER value;
8     VARS Node prev;
9     VARS Node next;
10 Node ENDSTRUCT /*Function declaration*/ FUNCTION BubbleSort(K, m)
11     VARS INTEGER i,j,head,head2,temp;
12     VARS INTEGER K_i,K_i_plus_1,K_m;
13     i = 0;
14     j = 0;
15     head = m;
16     WHILE (head > 1)
17         j = 0;
18         head2 = head - 1;
19         PRINT("Sorting elements 0 --> %d, ", head2);
20         FOR i:=0 TO 20 STEP 1
21             IF (K_i > K_i_plus_1) THEN
22                 temp = K_i_plus_1;
23                 K_i_plus_1 = K_i;
24                 K_i = temp;
25                 j = j + 1;
26             ENDIF
27         ENDFOR
28         PRINT("Element swaps in the current table scan: %d\n", j);
29         IF (j == 0) THEN
30             PRINT("No swap during last table scan. The table sorting completed successfully\n");
31             BREAK;
32         ENDIF
33         head = head - 1;
34     ENDWHILE
35 RETURN K_m END_FUNCTION
36 FUNCTION InsertionSort(K, m)
37     VARS INTEGER i,j,k,K_i,K_i_plus_1,K_j,K_m;
38     i = 0;
39     j = 0;
40     k = 0;
41     FOR j:=0 TO 20 STEP 1
42         PRINT("Sorting elements 0 --> %d\n", j);
43         k = K_j;
44         i = j - 1;
45         WHILE ((i > 0) AND (k < K_i))
46             K_i_plus_1 = K_i;
47             i = i - 1;
48         ENDWHILE
49     ENDFOR
50
```

Ακολουθεί στην επόμενη σελίδα το παράδειγμα ανεπιτυχούς εκτέλεσης.

ΠΑΡΑΔΕΙΓΜΑ ΑΝΕΠΙΤΥΧΟΥΣ ΕΚΤΕΛΕΣΗΣ:

Έστω ότι στη γραμμή 3 δεν έχει τοποθετηθεί η ακολουθία «*/» που δηλώνει το τέλος το σχολίου πολλαπλών γραμμών. Συνεπώς, ο αναλυτής εκλαμβάνει ως ένα σχόλιο την περιοχή από την αρχή του πρώτου σχόλιου μέχρι το τέλος του επόμενου, όπου σε αυτή την περιοχή περιλαμβάνονται και εντολές προγράμματος που δεν ανήκουν σε κάποιο σχόλιο.

Εμφανίζεται το αντίστοιχο μήνυμα σφάλματος στο τερματικό και η ανάλυση σταματάει, όπως φαίνεται στο παρακάτω στιγμιότυπο οθόνης:




```
takis@takis-VirtualBox: ~/parser
takis@takis-VirtualBox:~/parser$ bison -y -d c-like_parser.y
takis@takis-VirtualBox:~/parser$ flex c-like_lexer.l
takis@takis-VirtualBox:~/parser$ gcc -o myparser y.tab.c lex.yy.c -lfl
y.tab.c: In function 'yyparse':
y.tab.c:1596:16: warning: implicit declaration of function 'yylex' [-Wimplicit-function-declaration]
1596 | yychar = yylex ();
      |             ^
y.tab.c:1852:18: warning: passing argument 1 of 'yyerror' discards 'const' qualifier from pointer target type [-Wdiscarded-qualifiers]
1852 | yyerror (yymsp);
      |             ^
c-like_parser.y:9:14: note: expected 'char *' but argument is of type 'const char *'
9 | void yyerror(char *);
  |
takis@takis-VirtualBox:~/parser$ ./myparser prog.c
c-like parser, implemented by Christos-Panagiotis Mpatsouras, Student ID = 1054335
Program source code from parser input is following:

/* Created by Christos-Panagiotis Balatsouras, StudentID = 1054335
   CEID - University of Patras */
/*Program declaration
PROGRAM Sorting_Algorithms %This is a test program for lexer and parser development purposes
/*Struct declaration*/
ERROR: syntax error, unexpected TYPEDEF, expecting PROGRAM in line 6
TYPEDEFProgram Name: (null)
Lines of code scanned and parsed: 6

Variable List Items:
Function List Items:
takis@takis-VirtualBox:~/parser$
```

Ο κώδικας της εισόδου δεν έχει επιστραφεί ολόκληρος στην έξοδο, στο παραπάνω τερματικό και στο αρχείο output.c, καθώς έχει σταματήσει στη γραμμή που εμφανίζεται το σφάλμα.



```
1 /* Created by Christos-Panagiotis Balatsouras, StudentID = 1054335
2  CEID - University of Patras */
3 /*Program declaration
4 PROGRAM Sorting_Algorithms %This is a test program for lexer and parser development purposes
5 /*Struct declaration*/
6 TYPEDEF
```

5. ΠΑΡΑΡΤΗΜΑ – ΤΕΛΙΚΑ ΑΡΧΕΙΑ ΕΙΣΟΔΟΥ ΓΙΑ ΤΟ FLEX ΚΑΙ ΤΟ BISON

Σε αυτή την ενότητα παρουσιάζονται οι τελικές εκδόσεις των αρχείων περιγραφής της γλώσσας που δίνονται ως είσοδος στο Flex και στο Bison.

5.1 ΚΩΔΙΚΑΣ ΕΙΣΟΔΟΥ ΣΤΟ FLEX (ΑΡΧΕΙΟ ΕΙΣΟΔΟΥ c-like_lexer.l)

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

%option yylineno

underscore  "_"
digit       [0-9]
letter      [a-zA-Z]
chartype    "CHAR"
inttype     "INTEGER"
userdatatype "Node"
comparative ">"| "<"| "=="| "!="
char        .
charsequence {char}*
strliteral  \"{charsequence}\"
charliteral \"{char}\"
linecomment \"%{charsequence}
datatype    {chartype}|{inttype}
basiccharacter {underscore}|{letter}|{digit}
number      {digit}+|{digit}+\".\"{digit}+
basicstring  {basiccharacter}*
name        {letter}{basicstring}|{underscore}{basicstring}
array       \"{number}\"

%x MULTILINECOMM
%%

[ \t]+ { yylval.str = strdup(yytext); ECHO; printf(\"%s\",yytext);}
\"\\n\" { yylval.str = strdup(yytext); ECHO; printf(\"%s\",yytext); return NEWLINE; }
\";\"   { yylval.str = strdup(yytext); ECHO; printf(\"%s\",yytext); return ';'; }
\":\"   { yylval.str = strdup(yytext); ECHO; printf(\"%s\",yytext); return ':'; }
\", \"  { yylval.str = strdup(yytext); ECHO; printf(\"%s\",yytext); return ','; }
\"(\"    { yylval.str = strdup(yytext); ECHO; printf(\"%s\",yytext); return '('; }
\")\"    { yylval.str = strdup(yytext); ECHO; printf(\"%s\",yytext); return ')'; }
\"+\"    { yylval.str = strdup(yytext); ECHO; printf(\"%s\",yytext); return '+'; }
\"-\"    { yylval.str = strdup(yytext); ECHO; printf(\"%s\",yytext); return '-'; }
\"^\"    { yylval.str = strdup(yytext); ECHO; printf(\"%s\",yytext); return '^'; }
```

```
"*" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return '*'; }
"/" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return '/'; }
"=" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return '='; }
"AND" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return AND; }
"OR" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return OR; }
":=" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
ASSIGN_OPERATOR; }
"PROGRAM" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return PROGRAM; }
"VARS" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return VARS; }
"TYPEDEF" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return TYPEDEF; }
"STRUCT" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return STRUCT; }
"ENDSTRUCT" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
ENDSTRUCT; }
"FUNCTION" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
FUNCTION; }
"END_FUNCTION" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
END_FUNCTION; }
"RETURN" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return RETURN; }
"WHILE" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return WHILE; }
"ENDWHILE" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
ENDWHILE; }
"FOR" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return FOR; }
"TO" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return TO; }
"STEP" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return STEP; }
"ENDFOR" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return ENDFOR; }
"IF" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return IF; }
"THEN" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return THEN; }
"ELSEIF" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return ELSEIF; }
"ELSE" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return ELSE; }
"ENDIF" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return ENENDIF; }
"SWITCH" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return SWITCH; }
"CASE" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return CASE; }
"DEFAULT" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return DEFAULT;
}
"ENDSWITCH" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
ENDSWITCH; }
"PRINT" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return PRINT; }
"BREAK" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return BREAK; }
"STARTMAIN" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
STARTMAIN; }
"ENDMAIN" { yylval.str = strdup(yytext); ECHO; printf("%s\n",yytext); return
ENDMAIN; }
{strliteral} { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
STRLITERAL; }
{charliteral} { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
CHARLITERAL; }
{linecomment} { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); }
{userdatatype} { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return
USERDATATYPE; }
```

```
{datatype} { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return  
DATATYPE; }  
{name} { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return NAME; }  
{number} { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return NUM; }  
{array} { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return ARRAY; }  
{comparative} { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); return  
COMP_OPERATOR; }  
"/*" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext);  
BEGIN(MULTILINECOMM);}  
<MULTILINECOMM>[^*\n]* { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); }  
<MULTILINECOMM>"*" + [^*/\n]* { yylval.str = strdup(yytext); ECHO;  
printf("%s",yytext); }  
<MULTILINECOMM>"\n" { yylval.str = strdup(yytext); ECHO; printf("%s",yytext); }  
<MULTILINECOMM>"*" + "/" | "*" + "/" + "+" \n" { yylval.str = strdup(yytext); ECHO;  
printf("%s",yytext); BEGIN(INITIAL); }  
%%
```

Ακολουθεί στην επόμενη σελίδα ο κώδικας εισόδου στο Bison.

5.2 ΚΩΔΙΚΑΣ ΕΙΣΟΔΟΥ ΣΤΟ BISON (ΑΡΧΕΙΟ ΕΙΣΟΔΟΥ c-like_parser.y)

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

#define YYERROR_VERBOSE 1

void yyerror(char *);
extern FILE *yyin;
extern FILE *yyout;
extern int yylineno;
int line = 0;
FILE *diagnostics;
char *progr;

struct variableEntity
{
    char *Name;
    struct variableEntity *next;
};

struct functionEntity
{
    char *Name;
    struct functionEntity *next;
};

void printVariableList(struct variableEntity *node)
{
    while(node != NULL)
    {
        printf("Variable Name: %s\n", node->Name);
        node = node->next;
    }
}

void printFunctionList(struct functionEntity *node)
{
    while(node != NULL)
    {
        printf("Function Name: %s\n", node->Name);
        node = node->next;
    }
}

void variableListAppend(struct variableEntity** head_rf, char *name)
```

```
    struct variableEntity* new_var = (struct variableEntity*) malloc(sizeof(struct
variableEntity));
    struct variableEntity *end = *head_rf;

    new_var->Name = name;
    new_var->next = NULL;

    if(*head_rf == NULL)
    {
        *head_rf = new_var;
        return;
    }

    while(end->next != NULL)
    {
        end = end->next;
    }

    end->next = new_var;

    return;
}

void functionListAppend(struct functionEntity** head_rf, char *name)
{
    struct functionEntity* new_var = (struct functionEntity*) malloc(sizeof(struct
functionEntity));
    struct functionEntity *end = *head_rf;

    new_var->Name = name;
    new_var->next = NULL;

    if(*head_rf == NULL)
    {
        *head_rf = new_var;
        return;
    }

    while(end->next != NULL)
    {
        end = end->next;
    }

    end->next = new_var;

    return;
}

int variableSearch(struct variableEntity* head, char *key)
{

```

```
struct variableEntity* lookup = head;

while(lookup != NULL)
{
    if(strcmp(lookup->Name, key) == 0)
    {
        return 1;
    }
    lookup = lookup->next;
}
return 0;
}

int functionSearch(struct functionEntity* head, char *key)
{
    struct functionEntity* lookup = head;

    while(lookup != NULL)
    {
        if(strcmp(lookup->Name, key) == 0)
        {
            return 1;
        }
        lookup = lookup->next;
    }
    return 0;
}

struct variableEntity* variables = NULL;
struct functionEntity* functions = NULL;

%}

%union {
    char *str;
}

%token PROGRAM <str>NAME ARRAY NUM COMP_OPERATOR AND OR STRLITERAL CHARLITERAL
%token TYPEDEF STRUCT ENDSTRUCT
%token VARS DATATYPE USERDATATYPE FUNCTION END_FUNCTION RETURN
%token STARTMAIN ENDMAIN
%token WHILE ENDWHILE
%token FOR ASSIGN_OPERATOR TO STEP ENDFOR
%token IF THEN ELSEIF ELSE ENDIF
%token SWITCH CASE DEFAULT ENDSWITCH
%token PRINT
%token BREAK
%token NEWLINE
%left ','
%left '+' '-'
```

```
%left '*' '/'
%right '^'

%start program

%%

program: program_declaration main_statement newline { printf("Code Parsed
successfully!\n"); fprintf(diagnostics, "Code Parsed successfully!\n"); }
        | program_declaration function main_statement newline { printf("Code Parsed
successfully!\n"); fprintf(diagnostics, "Code Parsed successfully!\n"); }
        | program_declaration struct_statement main_statement newline { printf("Code
Parsed successfully!\n"); fprintf(diagnostics, "Code Parsed successfully!\n"); }
        | program_declaration struct_statement function main_statement newline {
printf("Code Parsed successfully!\n"); fprintf(diagnostics, "Code Parsed
successfully!\n"); }
        ;

program_declaration: PROGRAM NAME newline { progr = $2; }
                    ;

variable_declaration: VARS datatype declared_variables ';' newline
                    | variable_declaration VARS datatype declared_variables ';'
newline
                    ;

declared_variables: NAME { if(variableSearch(variables, $1) == 0){
variableListAppend(&variables, $1); } }
                    | NAME ARRAY { if(variableSearch(variables, $1) ==
0){variableListAppend(&variables, $1); } }
                    | declared_variables ',' declared_variables
                    ;

struct_statement: STRUCT NAME newline struct_variable ENDSTRUCT
                 | struct_statement STRUCT NAME newline struct_variable ENDSTRUCT
                 | TYPEDEF STRUCT USERDATATYPE newline struct_variable USERDATATYPE
ENDSTRUCT
                 | struct_statement TYPEDEF STRUCT USERDATATYPE newline
struct_variable USERDATATYPE ENDSTRUCT
                 ;

struct_variable: VARS datatype NAME ';' newline { if(variableSearch(variables, $3) ==
0){ variableListAppend(&variables, $3); } }
                | VARS datatype NAME ARRAY ';' newline { if(variableSearch(variables,
$3) == 0) { variableListAppend(&variables, $3); } }
                | struct_variable VARS datatype NAME ';' newline {
if(variableSearch(variables, $4) == 0) { variableListAppend(&variables, $4); } }
                | struct_variable VARS datatype NAME ARRAY ';' newline {
if(variableSearch(variables, $4) == 0) { variableListAppend(&variables, $4); } }
                ;
```



```
function: function_declaration commands function_end newline /*{ printf("Function
Statement found\n"); }*/
    | function_declaration variable_declaration commands function_end newline /*{
printf("Function Statement found\n"); }*/
    | function function_declaration commands newline function_end newline /*{
printf("Function Statement found\n"); }*/
    | function function_declaration variable_declaration commands function_end
newline /*{ printf("Function Statement found\n"); }*/
    ;

function_declaration: FUNCTION function_name newline /*{ printf("Function
Declared\n"); }*/
    ;

function_name: NAME '(' function_arguments ')' { if(functionSearch(functions, $1) ==
0) { functionListAppend(&functions, $1); } }
    ;

function_statement: NAME '(' variable ')' { if(functionSearch(functions, $1) == 0) {
printf("\nERROR: Function %s is NOT declared, in line %d\n", $1, yylineno);
fprintf(diagnostics, "ERROR: Function %s is NOT declared, in line %d\n", $1, yylineno);
YYABORT; } }
    ;

function_arguments: NAME { if(variableSearch(variables, $1) == 0) {
variableListAppend(&variables, $1); } }
    | function_arguments ',' function_arguments
    ;

function_end: RETURN NUM END_FUNCTION
    | RETURN CHARLITERAL END_FUNCTION
    | RETURN variable END_FUNCTION
    ;

main_statement: STARTMAIN commands ENDMAN /*{ printf("MAIN Statement found\n"); }*/
    | STARTMAIN variable_declaration commands ENDMAN /*{ printf("MAIN
Statement found\n"); }*/
    ;

command: assignment
    | print_statement
    | loop_statement
    | break_command
    | control_statement
    ;

commands: command newline
    | commands command newline
    ;
```

```
assignment: variable '=' expression ';'
           | assignment variable '=' expression ';'
           ;

loop_statement: while_statement
              | for_statement
              ;

break_command: BREAK ';' /*{ printf("Break command found\n"); }*/
              ;

control_statement: if_statement
                 | switch_statement
                 ;

while_statement: WHILE '(' condition ')' newline commands ENDWHILE /*{ printf("While
Loop statement found\n"); }*/
               ;

condition: expression COMP_OPERATOR expression
         | expression AND expression
         | expression OR expression
         | '(' condition ')' COMP_OPERATOR '(' condition ')'
         | '(' condition ')' AND '(' condition ')'
         | '(' condition ')' OR '(' condition ')'
         ;

for_statement: FOR NAME ASSIGN_OPERATOR NUM TO NUM STEP NUM newline commands ENDFOR {
if(variableSearch(variables, $<str>2) == 0) { printf("\nERROR: Variable %s is NOT
declared, in line %d\n",$<str>2,yylineno); fprintf(diagnostics,"ERROR: Variable %s is
NOT declared, in line %d\n",$<str>2,yylineno); YYABORT; } }
           ;

if_statement: IF '(' condition ')' THEN newline commands ENDIF /*{ printf("If
statement found\n"); }*/
            | IF '(' condition ')' THEN newline commands else_if_statement ENDIF /*{
printf("If statement found\n"); }*/
            | IF '(' condition ')' THEN newline commands else_statement ENDIF /*{
printf("If statement found\n"); }*/
            | IF '(' condition ')' THEN newline commands else_if_statement
else_statement ENDIF /*{ printf("If statement found\n"); }*/
            ;

else_if_statement: ELSEIF '(' condition ')' newline commands
                 | else_if_statement ELSEIF '(' condition ')' newline commands
                 ;

else_statement: ELSE newline commands
              ;
```

```
switch_statement: SWITCH '(' expression ')' newline case ENDSWITCH /*{ printf("Switch
statement found\n"); }*/
    | SWITCH '(' expression ')' newline case default ENDSWITCH /*{
printf("Switch statement found\n"); }*/
    ;

case: CASE '(' expression ')' ':' newline commands
    | case CASE '(' expression ')' ':' newline commands
    ;

default: DEFAULT ':' newline commands
    ;

print_statement: PRINT '(' STRLITERAL ')' ';' /*{ printf("Print Statement found\n");
}*/
    | PRINT '(' STRLITERAL ',' variable ')' ';' /*{ printf("Print
Statement found\n"); }*/
    ;

expression: literal
    | variable
    | function_statement
    | expression '+' expression
    | expression '-' expression
    | expression '^' expression
    | expression '*' expression
    | expression '/' expression
    | '(' expression ')'
    ;

variable: NAME { if(variableSearch(variables, $<str>1) == 0) { printf("\nERROR:
Variable %s is NOT declared, in line %d\n", $<str>1, yylineno);
fprintf(diagnostics, "ERROR: Variable %s is NOT declared, in line
%d\n", $<str>1, yylineno); YYABORT; } }
    | NAME ARRAY { if(variableSearch(variables, $<str>1) == 0) { printf("\nERROR:
Variable %s is NOT declared, in line %d\n", $<str>1, yylineno);
fprintf(diagnostics, "ERROR: Variable %s is NOT declared, in line
%d\n", $<str>1, yylineno); YYABORT; } }
    | variable ',' variable
    ;

datatype: DATATYPE
    | USERDATATYPE
    ;

literal: NUM
    | STRLITERAL
    | CHARLITERAL
    ;
```

```
newline: NEWLINE /*{ line++; }*/
    ;

%%

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

int main (int argc, char **argv)
{
    ++argv; --argc;
    if ( argc > 0 )
        yyin = fopen( argv[0], "r" );
    else
        yyin = stdin;
    yyout = fopen ( "output.c", "w" );

    printf("C-like parser, implemented by Christos-Panagiotis Mpalatsouras, Student
ID = 1054335\n");
    printf("Program source code from parser input is following: \n\n");

    diagnostics = fopen("diagnostics.txt", "w");
    fprintf(diagnostics, "**** START of Diagnostic Messages ****\n\n");

    yyparse();

    printf("Program Name: %s\n", progr);
    fprintf(diagnostics, "Program: %s\n\n", progr);
    printf("Lines of code scanned and parsed: %i\n", yylineno);
    fprintf(diagnostics, "Lines of code scanned and parsed: %i\n", yylineno);
    fprintf(diagnostics, "\n**** END of Diagnostic Messages ****\n");
    fclose(diagnostics);

    printf("\nVariable List Items: \n");
    printVariableList(variables);

    printf("\nFunction List Items: \n");
    printFunctionList(functions);

    free(variables);
    free(functions);

    return 0;
}
```

6. ΠΑΡΑΡΤΗΜΑ – ΤΕΛΙΚΟ ΑΡΧΕΙΟ ΕΙΣΟΔΟΥ ΣΤΟΝ PARSER ΓΙΑ ΕΛΕΓΧΟ ΤΗΣ ΛΕΙΤΟΥΡΓΙΑΣ ΤΟΥ

Σε αυτή την ενότητα παρουσιάζεται το αρχείο που δίνεται ως είσοδο στον parser για να ελεγχθεί η σωστή λειτουργία του.

```
/* Created by Christos-Panagiotis Balatsouras, StudentID = 1054335
   CEID - University of Patras */
/*Program declaration*/
PROGRAM Sorting_Algorithms %This is a test program for lexer and parser development
purposes
/*Struct declaration*/
TYPEDEF STRUCT Node
    VARS INTEGER value;
    VARS Node prev;
    VARS Node next;
Node ENDSTRUCT /*Function declaration*/ FUNCTION BubbleSort(K, m)
    VARS INTEGER i,j,head,head2,temp;
    VARS INTEGER K_i,K_i_plus_1,K_m;
    i = 0;
    j = 0;
    head = m;
    WHILE (head > 1)
        j = 0;
        head2 = head - 1;
        PRINT("Sorting elements 0 --> %d, ", head2);
        FOR i:=0 TO 20 STEP 1
            IF (K_i > K_i_plus_1) THEN
                temp = K_i_plus_1;
                K_i_plus_1 = K_i;
                K_i = temp;
                j = j + 1;
            ENDIF
        ENDFOR
        PRINT("Element swaps in the current table scan: %d\n", j);
        IF (j == 0) THEN
            PRINT("No swap during last table scan. The table sorting completed
successfully\n");
            BREAK;
        ENDIF
        head = head - 1;
    ENDWHILE
RETURN K_m END_FUNCTION
FUNCTION InsertionSort(K, m)
    VARS INTEGER i,j,k,K_i,K_i_plus_1,K_j,K_m;
    i = 0;
    j = 0;
    k = 0;
    FOR j:=2 TO 20 STEP 1
```

```
        PRINT("Sorting elements 0 --> %d\n", j);
        k = K_j;
        i = j - 1;
        WHILE ((i > 0) AND (k < K_i))
            K_i_plus_1 = K_i;
            i = i - 1;
        ENDWHILE
        K_i_plus_1 = k;
    ENDFOR
RETURN K_m END_FUNCTION
/*Main part of program*/
STARTMAIN VARS INTEGER n,i,c,A[20];
    VARS INTEGER values[20];
    VARS INTEGER option;
    VARS INTEGER A_n,A_i;
    n = 20;
    A[20] = values;
    PRINT("Initial Table: \n");
    FOR i:=0 TO 20 STEP 1
        PRINT("%d ", A_i);
    ENDFOR
    PRINT("\nOptions: \n");
    PRINT("1. Bubble-Sort:\n");
    PRINT("2. Insertion-Sort:\n");
    PRINT("3. Selection-Sort:\n");
    PRINT("Select Algorithm >> "); %scanf an option from keyboard
    SWITCH (option)
    CASE (1):
        A_n = BubbleSort(A,n);
        PRINT("Results: \n");
        FOR i:=0 TO 20 STEP 1
            PRINT("%d ", A_i);
        ENDFOR
        PRINT("\n");
    CASE (2):
        A_n = InsertionSort(A,n);
        PRINT("Results: \n");
        FOR i:=0 TO 20 STEP 1
            PRINT("%d ", A_i);
        ENDFOR
        PRINT("\n");
    DEFAULT:
        PRINT("No option selected\n");
    ENDSWITCH
ENDMAIN
```

7. ΒΙΒΛΙΟΓΡΑΦΙΑ

- [1] J. Levine, Flex & Bison: Text Processing Tools, O'Reilly Media, Inc., 2009.
- [2] «CS106X Handout #01 - 120 Introducing bison.pdf,» [Ηλεκτρονικό]. Available: <https://web.stanford.edu/class/archive/cs/cs143/cs143.1128/handouts/120%20Introducing%20bison.pdf>.
- [3] «CS106X Handout #01 - 050 Flex In A Nutshell.pdf,» [Ηλεκτρονικό]. Available: <https://web.stanford.edu/class/archive/cs/cs143/cs143.1128/handouts/050%20Flex%20In%20A%20Nutshell.pdf>.

Επιπλέον, για την υλοποίηση της εργασίας, λήφθηκαν υπ' όψη οι παρακάτω πηγές:

- Το manual του Flex και το manual του Bison από το e-class του μαθήματος «Αρχές γλωσσών προγραμματισμού και μεταφραστών».
- Οι διαφάνειες του μαθήματος «Αρχές γλωσσών προγραμματισμού και μεταφραστών» σχετικά με τα εργαλεία Flex και Bison.

ΤΕΛΟΣ ΤΕΚΜΗΡΙΩΣΗΣ