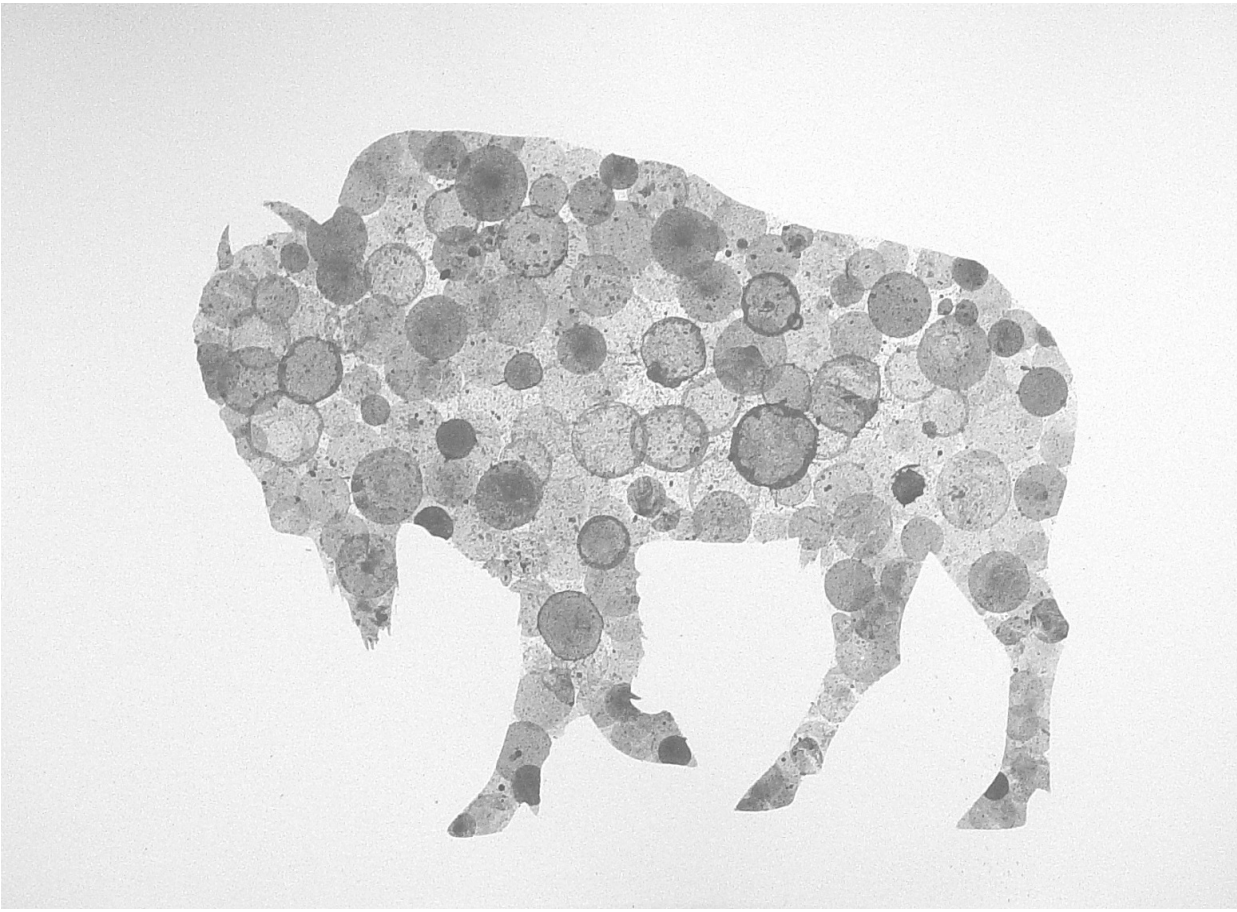
3873	Παπαρροδοπούλου Αναστασία	Καφφέζας Γιώργος	4465
4634	Σιαφλέκης Τάσος	Μαγριπής Άρης	4651



ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ · ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ & ΠΛΗΡΟΦΟΡΙΚΗΣ

ΑΡΧΕΣ ΓΛΩΣΣΩΝ
ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ & ΜΕΤΑΦΡΑΣΤΩΝ

ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ 2012-2013

1

ΘΕΩΡΗΤΙΚΟ ΤΜΗΜΑ

1.

Η γραμματική, για την οποία πρέπει να υπολογίσουμε τα σύνολα FIRST, FOLLOW και PREDICT, και να εξετάσουμε εάν είναι LL(1), παρατίθεται δίπλα, ενώ ο καθαυτός υπολογισμός των παραπάνω συνόλων γίνεται στη συνέχεια.

<S> ::= a<A> | b | <B>
<A> ::= <C>a | <D>b
<B> ::= <C>b | <D>a
<C> ::= <E>
<D> ::= <E>
<E> ::= ε

Σύνολα FIRST

FIRST(S) ≡ {a, b, ε}
FIRST(A) ≡ {ε}
FIRST(B) ≡ {ε}
FIRST(C) ≡ {ε}
FIRST(D) ≡ {ε}
FIRST(E) ≡ {ε}

Σύνολα FOLLOW

FOLLOW(S) ≡ {$$}
FOLLOW(A) ≡ {$$}
FOLLOW(B) ≡ {$$}
FOLLOW(C) ≡ {a, b}
FOLLOW(D) ≡ {a, b}
FOLLOW(E) ≡ {a, b}

Σύνολα PREDICT

PREDICT(S → aA) ≡ {a}
PREDICT(S → b) ≡ {b}
PREDICT(S → B) ≡ {$$}
PREDICT(A → Ca) ≡ {$$}
PREDICT(A → Db) ≡ {$$}
PREDICT(B → Cb) ≡ {$$}
PREDICT(B → Da) ≡ {$$}
PREDICT(C → E) ≡ {a, b}
PREDICT(D → E) ≡ {a, b}
PREDICT(E → ε) ≡ {a, b}

Πίνακας συντακτικής ανάλυσης

	a	b	$$
S	S → aA	S → b	S → B
A	-	-	A → Ca A → Db
B	-	-	B → Cb B → Da
C	C → E	C → E	-
D	D → E	D → E	-
E	E → ε	E → ε	-

Συμπέρασμα

Γνωρίζουμε ότι αν κάποια λεκτική μονάδα ανήκει στα σύνολα PREDICT περισσότερων από έναν κανόνων με το ίδιο αριστερό μέλος, τότε η γραμματική δεν είναι LL(1). Αυτό συμβαίνει στην περίπτωσή μας στον κανόνα A και B για το τερματικό σύμβολο $$. Έτσι, **η δοθείσα γλώσσα δεν είναι LL(1)**.

2.

Αγνοώντας αρχικά τη δοθείσα γραμματική και τα υπόλοιπα ζητούμενα της εκφώνησης, πρέπει πρώτα να απαντήσουμε στα δύο θεωρητικά ερωτήματα που διατυπώνονται. Ένα, για το τι ορίζεται ως σύγκρουση ολίσθησης-ελάττωσης, και δύο, πώς επιλύεται αυτή στα διάφορα είδη συντακτικών αναλυτών της οικογένειας LR. Θα αναφερθούμε και σε άλλα σημεία της θεωρίας που είναι απαραίτητα για την γενικότερη απάντηση στο παρόν ερώτημα.

Γνωρίζουμε ότι τα αρχικά **LR** σημαίνουν **«αριστερά προς τα δεξιά, δεξιότερη παραγωγή»** (left-to-right, right-most derivation) και αναφέρονται σε ανοδικούς (bottom-up) συντακτικούς αναλυτές. Αυτοί κατασκευάζουν το δένδρο συντακτικής ανάλυσης ξεκινώντας από τα φύλλα και προχωρώντας προς τη ρίζα, αναγνωρίζοντας πότε μια συλλογή φύλλων ή άλλων κόμβων μπορεί να συνενωθούν ως παιδιά ενός νέου κόμβου του δένδρου.

Οι αναλυτές αυτοί δε χρειάζεται ποτέ να προβλέψουν τι θα δουν στη συνέχεια. Αντ' αυτού, μεταφέρει λεκτικές μονάδες στο δάσος μέχρι να αναγνωρίσει το δεξιό μέλος κάποιου συντακτικού κανόνα, που αμέσως ελαττώνει στο αντίστοιχο αριστερό μέλος.

Πιο αναλυτικά, ένας τέτοιος αναλυτής θυμάται τις καταστάσεις από τις οποίες έχει περάσει τοποθετώντας τες στη στοίβα, μαζί με τα σύμβολα της γραμματικής. Στην πραγματικότητα, οι καταστάσεις (και όχι τα σύμβολα) είναι αυτές που καθοδηγούν τον αλγόριθμο συντακτικής ανάλυσης: μας λένε σε ποια κατάσταση ήμαστε στην αρχή ενός δεξιού μέλους. Συγκεκριμένα, όταν ο συνδυασμός κατάστασης και εισόδου μας λέει ότι χρειάζεται να ελαττώσουμε χρησιμοποιώντας τον κανόνα $A \rightarrow \alpha$, αφαιρούμε τόσα σύμβολα από τη στοίβα όσο το μήκος της συμβολοσειράς α, καθώς και το αρχείο των καταστάσεων στις οποίες βρεθήκαμε όταν ολισθαίναμε αυτά τα σύμβολα. Αυτές οι αφαιρέσεις φέρνουν πάλι στην επιφάνεια την κατάσταση στην οποία ήμαστε ακριβώς πριν από τις ολισθήσεις, επιτρέποντας έτσι να επιστρέψουμε σε αυτή την κατάσταση και να προχωρήσουμε σαν να είχαμε δει το A εξαρχής.

Όπως αναφέρεται και στο βιβλίο του M. L. Scott, μπορούμε να θεωρήσουμε τους κανόνες ολίσθησης ενός συντακτικού αναλυτή της οικογένειας LR ως τη συνάρτηση μετάβασης ενός πεπερασμένου αυτόματου, με μεγάλη ομοιότητα με τα αυτόματα που χρησιμοποιούνται για τη μοντελοποίηση των λεκτικών αναλυτών. Κάθε κατάσταση του αυτόματου αντιστοιχεί σε μία λίστα στοιχείων που δείχνουν πού μπορεί να βρίσκεται ο συντακτικός αναλυτής σε ένα συγκεκριμένο σημείο της συντακτικής ανάλυσης. Η μετάβαση για το σύμβολο εισόδου X (που μπορεί να είναι είτε τερματικό, είτε μη τερματικό) μας μεταφέρει στην κατάσταση της οποίας η βάση αποτελείται από τα στοιχεία στα οποία μετακινήθηκε η . μετά από ένα X στο δεξιό μέλος, συν όποια στοιχεία χρειάζεται να προστεθούν ως κλείσιμο. Ουσιαστικά η τελεία (.) στο δεξιό μέλος αναπαριστά τη θέση μας, δηλαδή τη θέση που αντιπροσωπεύει η κορυφή της στοίβας. Οι λίστες κατασκευάζονται από μια γεννήτρια ανοδικών συντακτικών αναλυτών προκειμένου να κατασκευαστεί το αυτόματο, αλλά δεν είναι απαραίτητες κατά τη διάρκεια της συντακτικής ανάλυσης.

Τα απλούστερα μέλη της οικογένειας των συντακτικών αναλυτών LR –οι LR(0), SLR(1), και LALR(1)– χρησιμοποιούν όλα το ίδιο αυτόματο, που λέγεται χαρακτηριστική μηχανή πεπερασμένης κατάστασης (characteristic finite-state machine) ή ΧΜΠΚ. Οι πλήρεις συντακτικοί αναλυτές LR χρησιμοποιούν μια μηχανή με αρκετά μεγαλύτερο πλήθος καταστάσεων (για τις περισσότερες γραμματικές). Οι διαφορές ανάμεσα στους δύο αυτούς αλγορίθμους εντοπίζονται στον τρόπο που χειρίζονται τις καταστάσεις που περιέχουν διενέξεις ή **συγκρούσεις ολίσθησης-ελάττωσης** (shift-reduce conflicts). Οι συγκρούσεις αυτές συμβαίνουν όταν υπάρχει ένα στοιχείο με την . στο μέσο (που σημαίνει την ανάγκη ολίσθησης) και ένα άλλο με την . στο τέλος (που σημαίνει την ανάγκη ελάττωσης).

Ένας συντακτικός αναλυτής **LR(0)** μπορεί να λειτουργήσει μόνο όταν δεν υπάρχουν τέτοιες καταστάσεις, καθώς δεν είναι σε θέση να τις επιλύσει χωρίς να κοιτάξει παρακάτω. Οι συντακτικοί αναλυτές **SLR** (simple LR) κοιτάζουν την επόμενη λεκτική μονάδα της εισόδου και χρησιμοποιούν τα σύνολα FOLLOW για να επιλύουν αυτές τις διενέξεις. Δηλαδή, ένας συντακτικός αναλυτής SLR θα επιλέξει την ελάττωση με τον κανόνα $A \rightarrow \alpha$ μόνο αν η επόμενη λεκτική μονάδα ανήκει επίσης στο σύνολο FIRST κάποιου από τα σύμβολα που ακολουθούν την . στα άλλα στοιχεία της κατάστασης. Φαίνεται ότι υπάρχουν σημαντικές περιπτώσεις στις οποίες μια λεκτική μονάδα μπορεί να ακολουθεί ένα συγκεκριμένο μη τερματικό σύμβολο κάπου μέσα στο πρόγραμμα, αλλά ποτέ σε πλαίσιο που περιγράφεται από την τρέχουσα κατάσταση. Για τις περιπτώσεις αυτές, τα καθολικά σύνολα FOLLOW δεν παρέχουν την απαιτούμενη ευελιξία.

Οι συντακτικοί αναλυτές **LALR** (look-ahead LR) αποτελούν βελτίωση των SLR επειδή χρησιμοποιούν τοπική προανάγνωση (εξαρτώμενη από την τρέχουσα κατάσταση), και είναι σε θέση να επιλύουν περισσότερες διενέξεις.. Διενέξεις ή συγκρούσεις μπορούν να προκύψουν και σε αυτούς, όταν το ίδιο σύνολο στοιχείων παρουσιαστεί σε δύο διαφορετικές διαδρομές της ΧΜΠΚ. Και οι δύο διαδρομές θα καταλήξουν στην ίδια κατάσταση, και σε εκείνο το σημείο η τοπική, εξαρτώμενη από την κατάσταση, προανάγνωση δε θα μπορεί να τις διακρίνει. Ένας **πλήρης συντακτικός αναλυτής LR** αναπαράγει τις καταστάσεις, ώστε να διατηρεί διακριτές τις διαδρομές όταν οι τοπικές πληροφορίες προανάγνωσης διαφέρουν.

Αφού εξετάσαμε τα θεωρητικά μέρη του ερωτήματος, θα περάσουμε στην κατασκευή της ΧΜΠΚ για τη δοθείσα γλώσσα (που παρατίθεται δίπλα), καθώς και τον πίνακα συντακτικής ανάλυσης SLR(1) που μας ζητείται. Για λόγους διακριτότητας και οικονομίας χώρου, ο συμβολισμός του μη τερματικού <A> στον πίνακα ισοδυναμεί με A, δηλαδή παραλείπονται τα <>.

<A> ::= y<B> | x | <B><C>
<B> ::= z<B> | u
<C> ::= s

	Κατάσταση	Μεταβάσεις
0.	A → •yB $$	*Με y ολίσθηση και μετάβαση στην 1.*
	A → •x $$	*Με x ολίσθηση και μετάβαση στην 2.*
	A → •BC $$	*Με B ολίσθηση και μετάβαση στην 3.*
	B → •zB	*Με z ολίσθηση και μετάβαση στην 6.*
	B → •u	*Με u ολίσθηση και ελάττωση (αφαίρεση μίας κατάστασης και τοποθέτηση B στην είσοδο).*
1.	A → y•B $$	*Με B ολίσθηση και μετάβαση στην 4.*
	B → •zB	*Με z ολίσθηση και μετάβαση στην 6.*
	B → •u	*Με u ολίσθηση και ελάττωση (αφαίρεση μίας κατάστασης και τοποθέτηση B στην είσοδο).*
2.	A → x • $$	*Με $$ ολίσθηση και ελάττωση (αφαίρεση δύο καταστάσεων και τοποθέτηση A στην είσοδο).*
3.	A → B•C $$	*Με C ολίσθηση και μετάβαση στην 5.*
	C → •s	*Με s ολίσθηση και ελάττωση (αφαίρεση μίας κατάστασης και τοποθέτηση C στην είσοδο).*
4.	A → yB • $$	*Με $$ ολίσθηση και ελάττωση (αφαίρεση τριών καταστάσεων και τοποθέτηση A στην είσοδο).*
5.	A → BC • $$	*Με $$ ολίσθηση και ελάττωση (αφαίρεση τριών καταστάσεων και τοποθέτηση A στην είσοδο).*
6.	B → z•B	*Με B ολίσθηση και ελάττωση (αφαίρεση δύο καταστάσεων και τοποθέτηση B στην είσοδο).*
	B → •zB	*Με z ολίσθηση και μετάβαση στην 6.*
	B → •u	*Με u ολίσθηση και ελάττωση (αφαίρεση μίας κατάστασης και τοποθέτηση B στην είσοδο).*

Για να φτιάξουμε και τον πίνακα συντακτικής ανάλυσης SLR(1), βοηθάει να γράψουμε τη γλώσσα μας διαχωρίζοντας τους συντακτικούς κανόνες:

1. A → yB
2. A → x
3. A → BC
4. B → zB
5. B → u
6. C → s

	x	y	z	u	s	$$
0.	s2	s1	s6	b5	-	-
1.	-	-	s6	b5	-	-
2.	-	-	-	-	-	b2
3.	-	-	-	-	b6	-
4.	-	-	-	-	-	b1
5.	-	-	-	-	-	b3
6.	-	-	s6	b5	-	-

Όπως και στο παράδειγμα του βιβλίου του Scott στη σελίδα 110, συμβολίζουμε την ολίσθηση με **s**, την ελάττωση με **r**, και τον συνδυασμό ολίσθησης και μετά ελάττωσης με **b**, ενώ ο αριθμός που ακολουθεί είναι η νέα κατάσταση, αν πρόκειται για ολίσθηση, ή ο κανόνας που χρησιμοποιείται, αν πρόκειται για (ολίσθηση και) ελάττωση.

Τώρα, για το ερώτημα του αν η δοθείσα γραμματική είναι **LR(0)**, θα λέγαμε πως **είναι**, καθώς παρατηρούμε ότι δεν έχει κανένα κενό συντακτικό κανόνα κι έτσι ο συντακτικός αναλυτής δε θα έχει κάποιο σχετικό πρόβλημα με την προανάγνωση του κενού. Για το ερώτημα του αν είναι **SLR(1)**, τέλος, θα λέγαμε ότι **είναι**, καθώς σε κάθε κατάσταση το τερματικό σύμβολο $$ εμφανίζεται σε μία ή καμία λεκτική μονάδα προανάγνωσης των στοιχείων της.

2
ΥΛΟΠΟΙΗΣΗ

Στο τμήμα αυτό παρατίθενται οι κώδικες για τα ερωτήματα της εργασίας που απαιτούνε υλοποίηση. Έγιναν τα δύο από τα τρία ερωτήματα που ζητούνταν, συγκεκριμένα το πρώτο και το δεύτερο. Ακολουθήθηκαν όσο ήταν δυνατό οι προδιαγραφές της γλώσσας που περιγράφονται στην εκφώνηση, αλλά μόνο σε ένα σημείο έγινε μία σύμβαση, για τους προσημασμένους αριθμούς. Αυτή έγινε ώστε να λυθούν μερικά προβλήματα αναγνώρισης του συν (+) και του πλην (-) ως προσήμων και όχι ως αριθμητικών τελεστών σε πρόσθεση και αφαίρεση αντίστοιχα. Η σύμβαση υπαγορεύει ότι ο προσημασμένος αριθμός (θετικός ή αρνητικός) πρέπει να βρίσκεται μέσα σε παρενθέσεις.

Να σημειωθεί ότι η εργασία έγινε σε Ubuntu 12.04 LTS και τα screenshots που παρατίθενται είναι σε ανάλυση 1024x600, λόγω μικρής οθόνης του netbook.

1. [80%]

Αρχείο Flex: 1/flex_file.l

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

%option noyywrap
%option yylineno

%x comments

sign ("+"|"-")
digit [0-9]
integer {digit}+
signed_integer {sign}{digit}+
character \'([^'"\\]|\\['"\\nt0])+\'
identifier [a-zA-Z_][a-zA-Z0-9_]*
white_spaces [ \t\n]

%%

"/*" BEGIN(comments);

<comments>{
	[^*\n]*
	"*"+[^*/\n]*
	\n
	"*/" BEGIN(INITIAL);
}

"if" {printf("\txxx KEYWORD: %s\n", yytext); return IF;}
"else" {printf("\txxx KEYWORD: %s\n", yytext); return ELSE;}
"integer" {printf("\txxx KEYWORD: %s\n", yytext); return INTEGER;}
```

```
"char" {printf("\txxx KEYWORD: %s\n", yytext); return CHAR;}
"void" {printf("\txxx KEYWORD: %s\n", yytext); return VOID;}
"class" {printf("\txxx KEYWORD: %s\n", yytext); return CLASS;}
"new" {printf("\txxx KEYWORD: %s\n", yytext); return NEW;}
"return" {printf("\txxx KEYWORD: %s\n", yytext); return RETURN;}
"while" {printf("\txxx KEYWORD: %s\n", yytext); return WHILE;}

"public" {printf("\txxx KEYWORD: %s\n", yytext); return PUBLIC;}
"protected" {printf("\txxx KEYWORD: %s\n", yytext); return PROTECTED;}
"private" {printf("\txxx KEYWORD: %s\n", yytext); return PRIVATE;}

"static" {printf("\txxx KEYWORD: %s\n", yytext); return STATIC;}
"abstract" {printf("\txxx KEYWORD: %s\n", yytext); return ABSTRACT;}
"final" {printf("\txxx KEYWORD: %s\n", yytext); return FINAL;}

"this" {printf("\txxx KEYWORD: %s\n", yytext); return THIS;}
"super" {printf("\txxx KEYWORD: %s\n", yytext); return SUPER;}

"+" {printf("\t||| SYMBOL: %s\n", yytext); return ADD;}
"-" {printf("\t||| SYMBOL: %s\n", yytext); return SUB;}
"*" {printf("\t||| SYMBOL: %s\n", yytext);return MUL;}
"/" {printf("\t||| SYMBOL: %s\n", yytext); return DIV;}
"%" {printf("\t||| SYMBOL: %s\n", yytext); return MOD;}

"==" {printf("\tooo XXXXXX %s\n", yytext); return EQ;}
"!=" {printf("\tooo XXXXXX %s\n", yytext); return DIF;}
">" {printf("\tooo XXXXXX %s\n", yytext); return GR;}
"<" {printf("\tooo XXXXXX %s\n", yytext); return LE;}
">=" {printf("\tooo XXXXXX %s\n", yytext); return GREQ;}
"<=" {printf("\tooo XXXXXX %s\n", yytext); return LEEQ;}

"||" {printf("\tooo YYYYYY %s\n", yytext); return OR;}
"&&" {printf("\tooo YYYYYY %s\n", yytext); return AND;}
"!" {printf("\tooo YYYYYY %s\n", yytext); return NOT;}

"(" {printf("\t... MMMMMM %s\n", yytext); return LEFT_PAR;}
")" {printf("\t... MMMMMM %s\n", yytext); return RIGHT_PAR;}
"{" {printf("\t... MMMMMM %s\n", yytext); return LEFT_C_BRA;}
"}" {printf("\t... MMMMMM %s\n", yytext); return RIGHT_C_BRA;}
"[" {printf("\t... MMMMMM %s\n", yytext); return LEFT_BRA;}
"]" {printf("\t... MMMMMM %s\n", yytext); return RIGHT_BRA;}

";" {printf("\t... MMMMMM %s\n", yytext); return DELIM;}
"=" {printf("\t... MMMMMM %s\n", yytext); return ASSIGN;}
"," {printf("\t... MMMMMM %s\n", yytext); return COMMA;}

{integer} {printf("\tvvv INTEGER: %s\n", yytext); return _INT;}
{signed_integer} {printf("\tvvv SIGNED INTEGER: %s\n", yytext); return _S_INT;}
{character} {printf("\twww CHARACTER: %s\n", yytext); return _CHAR;}
{identifier} {printf("\txxx IDENTIFIER: %s\n", yytext); return _IDENT;}
{white_spaces} {/* Ignore whitespaces. */}

%%
```

Αρχείο Bison: 1/bison_file.y

```
%{
```

```
        #include <stdio.h>
        #include <math.h>
        void yyerror(char *);
        extern FILE *yyin;
        extern FILE *yyout;
        extern yylineno;
        int errors;
%}

%token IF ELSE INTEGER CHAR VOID CLASS NEW RETURN WHILE
%token PUBLIC PROTECTED PRIVATE
%token STATIC ABSTRACT FINAL
%token THIS SUPER
%token ADD SUB MUL DIV MOD
%token EQ DIF GR LE GREQ LEEQ
%token OR AND NOT
%token _INT _S_INT _CHAR _IDENT
%token LEFT_PAR RIGHT_PAR LEFT_C_BRA RIGHT_C_BRA LEFT_BRA RIGHT_BRA
%token DELIM ASSIGN COMMA

%right ASSIGN
%left OR AND
%left EQ DIF GR LE GREQ LEEQ
%left ADD SUB
%left MUL DIV MOD
%right NOT

%%

/*****************************************************************************/

main_class: class_declaration
      | main_class class_declaration
      ;

class_declaration: CLASS _IDENT class_body {printf("\n------- VALID CLASS.\n\n");}
            ;

class_body: LEFT_C_BRA variable_declaration_area constructor_declaration method_declaration RIGHT_C_BRA
      ;

/*****************************************************************************/

variable_declaration_area: variable_declaration
                  | variable_declaration_area variable_declaration
                  ;

variable_declaration: type variable_declarators DELIM
                  | array_declarator DELIM
                  ;

type: INTEGER
   | CHAR
   ;

variable_declarators: variable_declarator
                  | variable_declarators COMMA variable_declarator
                  ;
```

```
variable_declarator: variable_declarator_name
            | variable_declarator_name ASSIGN variable_init
            ;

variable_declarator_name: _IDENT
                ;

variable_init: expression
        | _CHAR
        ;

array_declarator: array_declarator_name ASSIGN NEW type LEFT_BRA _INT RIGHT_BRA
          ;

array_declarator_name: _IDENT
              ;

/*****************************************************************************/

constructor_declaration: constructor_modifier constructor_declarator constructor_body
;

constructor_modifier: PUBLIC
              | PROTECTED
              | PRIVATE
              ;

constructor_declarator: constructor_name LEFT_PAR parameter_list RIGHT_PAR
               ;

constructor_name: _IDENT
          ;

parameter_list: parameter
         | parameter_list COMMA parameter
         ;

parameter: /* empty */
      | type variable_declarator_name
      ;

constructor_body: LEFT_C_BRA explicit_constructor_invocation block_statements RIGHT_C_BRA
          | LEFT_C_BRA explicit_constructor_invocation RIGHT_C_BRA /* check all cases */
          ;

explicit_constructor_invocation: /* empty */
                    | THIS LEFT_PAR explicit_parameter_list RIGHT_PAR DELIM
                    | SUPER LEFT_PAR explicit_parameter_list RIGHT_PAR DELIM
                    ;

explicit_parameter_list: explicit_parameter
                | explicit_parameter_list COMMA explicit_parameter
                ;

explicit_parameter: /* empty */
            | _IDENT
            ;

/*****************************************************************************/
```

```
method_declaration: method_header method_body
;

method_header: method_modifiers result_type method_declarator
;

method_modifiers: /* empty */
            | method_modifier
            | method_modifiers COMMA method_modifier
            ;

method_modifier: PUBLIC
            | PROTECTED
            | PRIVATE
            | STATIC
            | ABSTRACT
            | FINAL
            ;

result_type: type
        | VOID
        ;

method_declarator: method_name LEFT_PAR parameter_list RIGHT_PAR
             ;

method_name: _IDENT
        ;

method_body: LEFT_C_BRA variable_declaration block_statements RIGHT_C_BRA
        | LEFT_C_BRA variable_declaration RIGHT_C_BRA
        | DELIM
        ;

/*****************************************************************************/

block: LEFT_C_BRA block_statements RIGHT_C_BRA
    ;

block_statements: block_statement
            | block_statements block_statement
            ;

block_statement: local_variable_declaration
            | statement
            ;

local_variable_declaration: type variable_declarators DELIM
                    ;

statement: if_statement
       | if_else_statement
       | while_statement
       | statement_without_trailing
       ;

statement_no_short: if_else_statement_no_short
             | while_statement_no_short
```

```
            | statement_without_trailing
            ;

statement_without_trailing: block
                  | empty_statement
                  | expression_statement
                  | return_statement
                  ;

empty_statement: DELIM
          ;

expression_statement: expression
              ;

if_statement: IF LEFT_PAR conditional_expression RIGHT_PAR statement
        ;

if_else_statement: IF LEFT_PAR conditional_expression RIGHT_PAR statement_no_short ELSE statement
            ;

if_else_statement_no_short: IF LEFT_PAR conditional_expression RIGHT_PAR statement_no_short ELSE statement_no_short
                  ;

while_statement: WHILE LEFT_PAR conditional_expression RIGHT_PAR statement
          ;

while_statement_no_short: WHILE LEFT_PAR conditional_expression RIGHT_PAR statement_no_short
                ;

return_statement: RETURN expression
           ;

/*****************************************************************************/

expression: number
       | _IDENT
       | LEFT_PAR expression RIGHT_PAR
       | assignment_expression
       | numerical_expression
       | logical_expression
       | array_expression
       ;

number: _INT
    | LEFT_PAR _S_INT RIGHT_PAR
    ;

conditional_expression: /* empty */
                | expression
                | relational_expression
                ;

assignment_expression: expression ASSIGN expression
                ;

numerical_expression: expression ADD expression
                | expression SUB expression
                | expression MUL expression
```

```
                | expression DIV expression
                | expression MOD expression
                ;

relational_expression: expression EQ expression
                | expression DIF expression
                | expression GR expression
                | expression LE expression
                | expression GREQ expression
                | expression LEEQ expression
                ;

logical_expression: expression OR expression
                | expression AND expression
                | NOT expression
                ;

array_expression: _IDENT LEFT_BRA _INT RIGHT_BRA
                ;

/*****************************************************************************/

%%

void yyerror(char *s){
    errors++;
        printf("\n------- ERROR AT LINE #%d.\n\n", yylineno);
}

int main (int argc, char **argv){
        argv++;
        argc--;
        errors=0;

        if(argc>0)
                yyin=fopen(argv[0], "r");
        else
                yyin=stdin;

        yyparse();

        if(errors==0)
            printf("------- PARSING COMPLETED, VALID PROGRAM.\n\n");

        return 0;
}
```

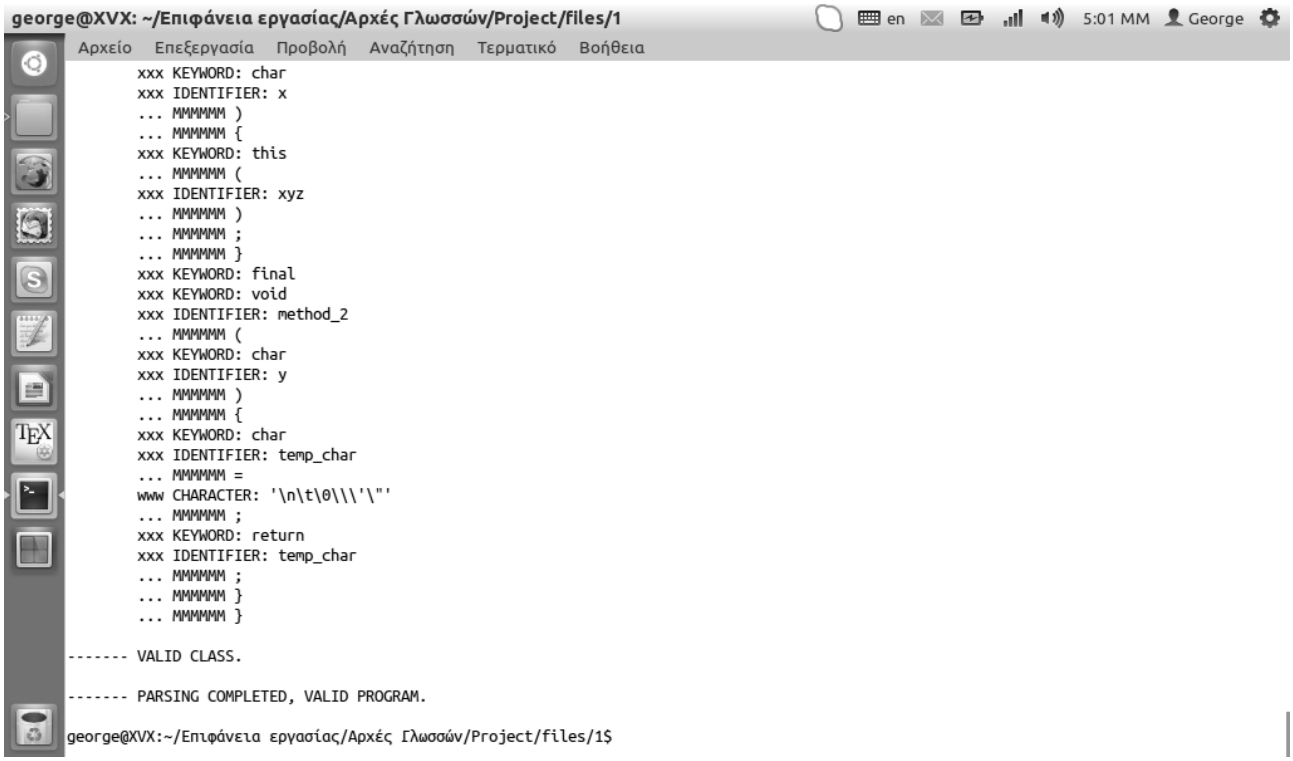
Για την εκτέλεση των παραπάνω αρχείων απαιτείται το compile τους κι η εκτέλεση με τις εξής εντολές:

- bison -y -d bison_file.y
- flex flex_file.l
- gcc -c y.tab.c lex.yy.c
- gcc y.tab.o lex.yy.o -o parser
- ./parser input_file.txt

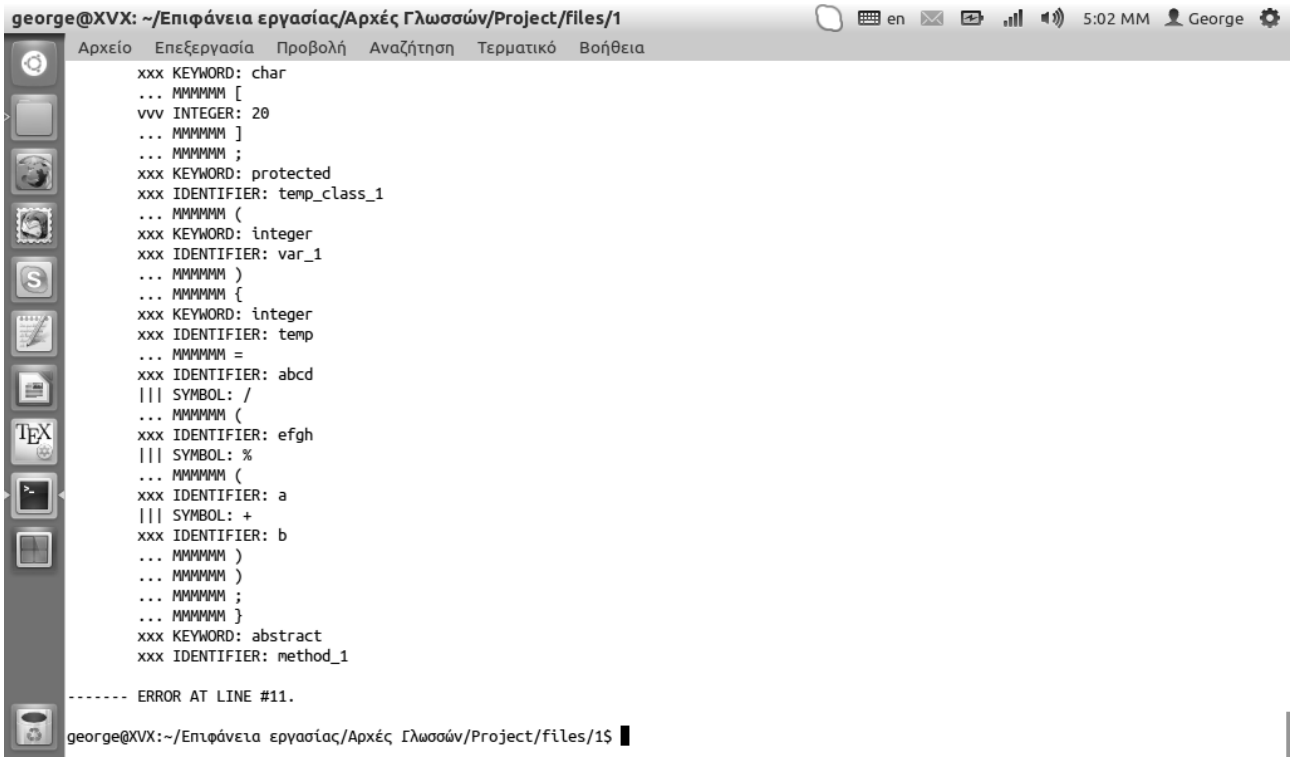
Στο ερώτημα ζητείται να το τρέξουμε μία φορά για ένα πρόγραμμα συντακτικά ορθό, και άλλη μία για λάθος, ώστε να εμφανίζονται τα αντίστοιχα μηνύματα αποδοχής ή απόρριψης από το λεκτικό και συντακτικό αναλυτή. Έτσι,

έχουμε τα ακόλουθα αποτελέσματα:

α) για το **αποδεκτό** πρόγραμμα 1/input_file.txt, του οποίου τα αποτελέσματα βρίσκονται πλήρη στο 1/output_file.txt.



β) για το **μη-αποδεκτό** πρόγραμμα 1/wrong_input_file.txt, με τα πλήρη αποτελέσματα στο 1/wrong_output_file.txt.



Εδώ εμφανίζεται και η ένδειξη του λάθους μαζί με την γραμμή στην οποία εντοπίστηκε (στη γραμμή 11).

2. [10%]

Αρχείο Flex: 2/flex_file.l

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

%option noyywrap
%option yylineno

%x comments
%x include_directive

sign ("+"|"-")
digit [0-9]
integer {digit}+
signed_integer {sign}{digit}+
character \'([^'"\\]|\\['"\\nt0])+\'
identifier [a-zA-Z_][a-zA-Z0-9_]*
white_spaces [ \t\n]

%%

#include BEGIN(include_directive);

"/*" BEGIN(comments);

<comments>{
   [^*\n]*
   "*"+[^*/\n]*
   \n
   "*/" BEGIN(INITIAL);
}

"if" {printf("\txxx KEYWORD: %s\n", yytext); return IF;}
"else" {printf("\txxx KEYWORD: %s\n", yytext); return ELSE;}
"integer" {printf("\txxx KEYWORD: %s\n", yytext); return INTEGER;}
"char" {printf("\txxx KEYWORD: %s\n", yytext); return CHAR;}
"void" {printf("\txxx KEYWORD: %s\n", yytext); return VOID;}
"class" {printf("\txxx KEYWORD: %s\n", yytext); return CLASS;}
"new" {printf("\txxx KEYWORD: %s\n", yytext); return NEW;}
"return" {printf("\txxx KEYWORD: %s\n", yytext); return RETURN;}
"while" {printf("\txxx KEYWORD: %s\n", yytext); return WHILE;}

"public" {printf("\txxx KEYWORD: %s\n", yytext); return PUBLIC;}
"protected" {printf("\txxx KEYWORD: %s\n", yytext); return PROTECTED;}
"private" {printf("\txxx KEYWORD: %s\n", yytext); return PRIVATE;}

"static" {printf("\txxx KEYWORD: %s\n", yytext); return STATIC;}
"abstract" {printf("\txxx KEYWORD: %s\n", yytext); return ABSTRACT;}
"final" {printf("\txxx KEYWORD: %s\n", yytext); return FINAL;}

"this" {printf("\txxx KEYWORD: %s\n", yytext); return THIS;}
"super" {printf("\txxx KEYWORD: %s\n", yytext); return SUPER;}
```

```
"+" {printf("\t||| SYMBOL: %s\n", yytext); return ADD;}
"-" {printf("\t||| SYMBOL: %s\n", yytext); return SUB;}
"*" {printf("\t||| SYMBOL: %s\n", yytext);return MUL;}
"/" {printf("\t||| SYMBOL: %s\n", yytext); return DIV;}
"%" {printf("\t||| SYMBOL: %s\n", yytext); return MOD;}

"==" {printf("\tooo XXXXXX %s\n", yytext); return EQ;}
"!=" {printf("\tooo XXXXXX %s\n", yytext); return DIF;}
">" {printf("\tooo XXXXXX %s\n", yytext); return GR;}
"<" {printf("\tooo XXXXXX %s\n", yytext); return LE;}
">=" {printf("\tooo XXXXXX %s\n", yytext); return GREQ;}
"<=" {printf("\tooo XXXXXX %s\n", yytext); return LEEQ;}

"||" {printf("\tooo YYYYYY %s\n", yytext); return OR;}
"&&" {printf("\tooo YYYYYY %s\n", yytext); return AND;}
"!" {printf("\tooo YYYYYY %s\n", yytext); return NOT;}

"(" {printf("\t... MMMMMM %s\n", yytext); return LEFT_PAR;}
")" {printf("\t... MMMMMM %s\n", yytext); return RIGHT_PAR;}
"{" {printf("\t... MMMMMM %s\n", yytext); return LEFT_C_BRA;}
"}" {printf("\t... MMMMMM %s\n", yytext); return RIGHT_C_BRA;}
"[" {printf("\t... MMMMMM %s\n", yytext); return LEFT_BRA;}
"]" {printf("\t... MMMMMM %s\n", yytext); return RIGHT_BRA;}

";" {printf("\t... MMMMMM %s\n", yytext); return DELIM;}
"=" {printf("\t... MMMMMM %s\n", yytext); return ASSIGN;}
"," {printf("\t... MMMMMM %s\n", yytext); return COMMA;}

{integer} {printf("\tvvv INTEGER: %s\n", yytext); return _INT;}
{signed_integer} {printf("\tvvv SIGNED INTEGER: %s\n", yytext); return _S_INT;}
{character} {printf("\twww CHARACTER: %s\n", yytext); return _CHAR;}
{identifier} {printf("\txxx IDENTIFIER: %s\n", yytext); return _IDENT;}
{white_spaces} {/* Ignore whitespaces. */}

<include_directive>[ \t]*     /* Ignore whitespaces. */
<include_directive>[^ \t\n]+ { /* Getting filename. */
    yyin=fopen(yytext, "r");

    if(!yyin)
        printf("\t\tERROR: could not include the selected file.");

    yypush_buffer_state(yy_create_buffer(yyin, YY_BUF_SIZE));

    BEGIN(INITIAL);
}

<<EOF>> {
    yypop_buffer_state();

    if(!YY_CURRENT_BUFFER){
        yyterminate();
    }
}

%%
```

Αρχείο Bison: 2/bison_file.y.

(το αρχείο Bison είναι ίδιο με αυτό του προηγούμενου ερωτήματος, γι' αυτό και δεν παρατίθεται εκ νέου)

Για την επιβεβαίωση της ορθής λειτουργίας των παραπάνω, ακολουθεί το παρακάτω screenshot για το αρχείο 2/input_file.txt με συμπεριλαμβανόμενο το αρχείο 2/included_file.txt, ενώ τα αποτελέσματα υπάρχουν και στο αρχείο 2/output_file.txt.