13/5/2017

Πολυτεχνείο Κρήτης

Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Θεωρία Υπολογισμού

(ΠΛΗ 402)

Εαρινό Εξάμηνο 2016-2017

Αναφορά Εργασίας Προγραμματισμού

Λεκτική και Συντακτική Ανάλυση της Γλώσσας Προγραμματισμού FC

Στοιχεία ομάδας

Ομάδα:

Αριθμός Ομάδας: LAB40233128

Ονοματεπώνυμο: *Καμπυλαυκάς Αναστάσιος*

Α.Μ.: *2012030143*

Σκοπός της εργασίας

Η εργασία προγραμματισμού του μαθήματος **«ΠΛΗ 402 – Θεωρία Υπολογισμού»** έχει ως στόχο την βαθύτερη κατανόηση της χρήσης και εφαρμογής θεωρητικών εργαλείων, όπως οι **κανονικές εκφράσεις και οι γραμματικές χωρίς συμφραζόμενα**, στο πρόβλημα της μεταγλώττισης (compilation) γλωσσών προγραμματισμού. Συγκεκριμένα, η εργασία αφορά στην **σχεδίαση και υλοποίηση των αρχικών σταδίων ενός μεταγλωττιστή** (compiler) για την φανταστική γλώσσα προγραμματισμού **FC** (Fictional C).

Εργασία προγραμματισμού

Το τελικό προϊόν της εργασίας είναι να η δημιουργία ενός **source-to-source compiler (trans-compiler ή transpiler)**, δηλαδή ενός τύπου μεταγλωττιστή ο οποίος παίρνει ως είσοδο τον πηγαίο κώδικα ένας προγράμματος σε μια γλώσσα προγραμματισμού και παράγει τον ισοδύναμο πηγαίο κώδικα σε μια άλλη γλώσσα προγραμματισμού. Στην περίπτωσή μας ο πηγαίος κώδικας εισόδου θα είναι γραμμένος στη φανταστική γλώσσα προγραμματισμού **FC** και ο παραγόμενος κώδικας θα είναι στη γλώσσα προγραμματισμού **C**.

Η εργασία περιλαμβάνει τα εξής δύο στάδια:

- Υλοποίηση λεκτικού αναλυτή για τη γλώσσα FC με χρήση ***flex***.
- Υλοποίηση συντακτικού αναλυτή για τη γλώσσα FC με χρήση ***bison***.

Για την λεκτική ανάλυση με την βοήθεια του εργαλείου ***flex*** κατασκευάστηκε ένα πρόγραμμα **(FC_lexer.l)** που παίρνει ως είσοδο ένα πρόγραμμα γραμμένο στην γλώσσα προγραμματισμού FC και αναγνωρίζει τις λεκτικές μονάδες του.

Για την συντακτική ανάλυση με τη βοήθεια του εργαλείου ***bison*** κατασκευάστηκε ένα νέο πρόγραμμα **(FC_pasrser.y)** που παίρνει ως είσοδο ένα πρόγραμμα σε FC και σε συνεργασία με τον λεκτικό αναλυτή (FC_lexer.l) αποφαίνεται για το αν ακολουθεί τους συντακτικούς κανόνες της FC ή όχι.

Λεκτική ανάλυση με το εργαλείο Flex

Γενικά στοιχεία

Ο λεκτικός αναλυτής διαβάζει ένα πρόγραμμα εισόδου σε FC και προσπαθεί να εντοπίσει patterns (κανονικές εκφράσεις) που έχουν δηλωθεί στο flex (FC_lexer.l αρχείο). Αν ταιριάζουν δύο η περισσότερα patterns επιλέγει αυτό με το μεγαλύτερο μήκος. Αν το μήκος είναι ίδιο επιλέγει το pattern που δηλώθηκε πρώτο στο αρχείο flex (προτεραιότητα από πάνω προς τα κάτω στον flex). Κάθε φορά που γίνεται ένα «ταίριασμα» εκτελείται το αντίστοιχο action που καθορίζει το κάθε pattern. Το action στην περίπτωσή μας είναι να επιστρέφει ο flex το όνομα του token που έγινε ταίριασμα (π.χ. **{return KW_BEGIN;}**). Το return αυτό που κάνει ο flex θα χρησιμοποιηθεί ως είσοδο στον συντακτικό αναλυτή (Bison) στη συνέχεια (βλέπε παρακάτω).

Definitions Section

Όσον αφορά το περιεχόμενο του λεκτικού αναλυτή που υλοποιήθηκε, στο Rules Section αυτού δηλώθηκαν όλα τα σύνθετα tokens που ζητήθηκαν από την εκφώνηση της άσκησης. Σε αυτό το κομμάτι του προγράμματος δηλαδή έγινε ο ορισμός όλων των σύνθετων tokens τα οποία θα χρησιμοποιηθούν στη συνέχεια, στο Rules Section του προγράμματος.

Rules Section

Ο κανόνας ***(?-i:**)***, όπου ** οποιοδήποτε Keyword (π.χ. if, integer, κτλ.) δηλώνει ένα **case sensitive (-i) token**.

Ο κανόνας *"*****"* όπου * οποιοσδήποτε ANSI χαρακτήρας δηλώνει τον αντίστοιχο χαρακτήρα. **(π.χ. \(δηλώνει (, το \& δηλώνει &, το \\ δηλώνει \, κτλ.)**

Ο κανόνας "***%x comment***" που χρησιμοποιείται για τους κανόνες των multiple line comment ορίζει ένα **exclusive start condition**, το οποίο ενεργοποιείται όταν ο flex δεχθεί ως είσοδο τους χαρακτήρες "***/****" ("***/*** **BEGIN(comment);"**). Κάθε exclusive start condition περιέχει patterns με πρόθεμα-αρχική συνθήκη **< ** >** (π.χ. ***<comment>*** στον συγκεκριμένο lexer) τα οποία ενεργοποιούνται μόνο όταν είναι ενεργοί οι αντίστοιχοι κανόνες των comment. Όταν είναι ενεργοί οι κανόνες των comment κανένα άλλο pattern του flex δεν είναι ενεργοποιημένο

εκτός από τα pattern που έχουν το πρόθεμα "<comment>". Όταν ο flex πάρει ως είσοδο τους χαρακτήρες "*/" που θα σημάνουν το τέλος των σχολίων πολλαπλών γραμμών ξαναενεργοποιούνται όλα τα patterns του flex μέσω της εντολής "BEGIN(INITIAL);", όπου initial είναι η αρχική κατάσταση, όπου μόνο οι κανόνες χωρίς αρχικές συνθήκες (π.χ. <comment>) είναι ενεργοί.

Στους κανόνες {ID} {NUMBER} και {REAL} μας ενδιαφέρει εκτός από το γεγονός της αναγνώρισης των συγκεκριμένων κανόνων και η αντίστοιχη συμβολοσειρά ή ο αριθμός που ανιχνεύθηκε. Είναι λοιπόν αναγκαίο να υπάρχει ένας στοιχειώδης διαχωρισμός των τιμών αυτών των τιμών επιστροφής ούτως ώστε να ξέρει ο Bison πως να τις διαχειριστεί. Γι΄ αυτόν ακριβώς τον λόγο χρησιμοποιείται ένα union μέσα στον κώδικα του συντακτικού αναλυτή που περιλαμβάνει όλα τα διαφορετικά είδη μεταβλητών που μπορεί να χρησιμοποιηθούν. Παρατίθεται η προαναφερθείσα Union.

***** Bison *****

```
%union
{
  char* str;
  int intNum;
  double doubleNum;
}
```

***** Flex *****

```
{ID}          { yylval.str = strdup(yytext); return IDENT; }
{NUMBER}      { yylval.intNum = atoi(yytext); return POSINT; }
{REAL}        { yylval.doubleNum = atof(yytext); return REAL; }
```

Με τον παραπάνω τρόπο λοιπόν και χρησιμοποιώντας την global μεταβλητή yyval είναι δυνατό στον λεκτικό αναλυτή να προσδιοριστεί ο τύπος των αντίστοιχων μεταβλητών. Η μεταβλητή yytext περιέχει τη αντίστοιχη τιμή των μεταβλητών.

Π.χ.	iAmTiredOfThisReport	{ID}
	56	{NUMBER}
	45.65e+32	{REAL}

Παραπάνω χρησιμοποιήθηκαν οι εντολές atoi και atof για την επιστροφή ακέραιων και δεκαδικών αριθμών αντίστοιχα.

- {yylval.intNum = atoi(yytext); return POSINT;} για τους ακέραιους
- {yylval.doubleNum = atof(yytext); return REAL;} για τους δεκαδικούς

Η χρήση αυτών των εντολών έχει δύο οφέλη, αφενός επιστρέφουν **integer** η **atoi** και **double** η **atof** ούτως ώστε να συμπίπτουν οι τιμές επιστροφής του lexer με αυτές που περιμένει ο Bison (όπως αναφέρθηκε παραπάνω) και αφετέρου κάνουν eliminate τα περιττά μηδενικά στην αρχή των αριθμών. Η atof κάνει eliminate και τα περιττά μηδενικά μετά τον τελευταίο αριθμό του αντίστοιχου αριθμού μετά την υποδιαστολή.

Π.χ. **(atoi)** 004500 -> 4500

(atof) 006500.004300000 -> 6500.0043

Όσον αφορά τα σχόλια, τόσο τα σχόλια γραμμής όσο και τα σχόλια πολλαπλών γραμμών, αυτά περιγράφονται από patterns τα οποία δεν επιστρέφουν τίποτα στον Bison, χρησιμοποιούνται **μόνο για να αναγνωρίζουν ολόκληρα τα αντίστοιχα σχόλια και να τα κάνουν eliminate**. Όπου χρειάζεται γίνεται αύξηση του αριθμού γραμμής (lineNum++), ούτως ώστε να συμβαδίζει η γραμμή στην οποία βρίσκεται κάποια «λέξη» (token) στον κώδικα που μας δίνεται, στο αρχείο εισόδου *.fc, με την γραμμή στην οποία δηλώνουμε ότι εμφανίζεται στην λεκτική ανάλυση που κάνουμε. Αυτό έγινε για να μην «χάνεται» η αρίθμηση των σειρών από τις γραμμές που περιέχουν comments, αλλά να συνυπολογίζονται κανονικά στις γραμμές, αν και δεν εμφανίζονται στην λεκτική ανάλυση ούτε επιστρέφονται στον συντακτικό αναλυτή (βλέπε παρακάτω).

Ο τελευταίος κανόνας:

```
. { yyerror("\nLexical Error: Unrecognized literal \"%s\" in line: %d\n", yytext, lineNum); return EOF;}
```

Αναγνωρίζεται μόνο όταν ο λεκτικός αναλυτής δεν ταιριάξει το σύμβολο που βρήκε στη συμβολοσειρά εισόδου (αρχείο input.fc) με κάποιο από τα tokens του, που αποτελεί και ολόκληρο το λεξιλόγιο της γλώσσας FC και γι’ αυτό το λόγο **εμφανίζει και ένα μήνυμα λεξικού λάθους** όποτε αναγνωριστεί.

Επιπλέον στοιχεία για τον λεκτικό αναλυτή

Ο λεκτικός αναλυτής κάνει include την βιβλιοθήκη **FC_parser.tab.h** η οποία παράγεται από την εντολή "**bison *-d* -v -r all --report=solved FC_parser.y**". Μέσα σε αυτή τη βιβλιοθήκη είναι ορισμένα όλα τα tokens (π.χ. KW_STATIC) τα οποία έχουν γίνει defined (π.χ. "%token KW_STATIK") στον συντακτικό αναλυτή και τα οποία επιστρέφει ο λεκτικός αναλυτής στον συντακτικό (αναλυτή) κατά την συνεργασία τους, όταν αναγνωρίσει κάποιο από αυτά (όπως αναφέρθηκε και παραπάνω).

Π.χ. *"(?-i:integer) { return KW_INTEGER;}"*

Χρησιμοποιήθηκαν επίσης οι εντολές **%nounput και %noinput**, οι οποίες υποδεικνύουν στον flex να ***μην*** δημιουργήσει τις αντίστοιχες συναρτήσεις (**yyunput και input**) για να μην εμφανίζονται τα **warnings** που έβγαζε ο gcc γι' αυτές ότι αρχικοποιούνται στο αρχείο FC_lexer.c (από την εντολή flex FC_lexer.l) αλλά δεν χρησιμοποιούνται πουθενά μέσα στο πρόγραμμά.

Συντακτική ανάλυση με το εργαλείο Bison

Γενικά στοιχεία

Σε ένα αυτόματο στοίβας ένας συντακτικός κανόνας είναι της μορφής:

Non_terminal_symbol → Non_terminal_symbol Terminal_symbol...

Όπου το αριστερό μέρος αποτελείται από ένα **μη τερματικό σύμβολο**, ενώ τα σύμβολα στο δεξιό μέρος του κανόνα μπορεί να είναι **είτε τερματικά είτε μη τερματικά** (αριστερό μέρος άλλου γραμματικού κανόνα). Ο συντακτικός αναλυτής δέχεται ως είσοδο την έξοδο του lexer (FC_lexer.l) που αποτελείται από τα tokens και προσπαθεί μέσω των συντακτικών κανόνων του να αποφανθεί για το αν το πρόγραμμα εισόδου ικανοποιεί τους συντακτικούς κανόνες της FC ή όχι.

Ο συντακτικός αναλυτής του Bison λειτουργεί **Bottom–up** καθώς προσπαθεί μέσω πολλαπλών shift και reduce να κάνει reduce ολόκληρη την είσοδο στο start symbol. Ουσιαστικά διαβάζει ένα-ένα τα σύμβολα και τα εισάγει σε μία στοίβα (**λειτουργία Shift**). Όταν μια ακολουθία από σύμβολα ικανοποιεί έναν γραμματικό κανόνα του, τα σύμβολα βγαίνουν από την στοίβα και μπαίνει σε αυτή το αρίστερο σύμβολο (τερματικό) του συντακτικού κανόνα (**λειτουργία Reduce**). Η διαδικασία ολοκληρώνεται επιτυχώς αν γίνει reduce το start symbol και τελειώσει παράλληλα και η είσοδος (το αρχείο εισόδου φτάσει στο <<EOF>>).

Σκοπός του parser είναι να είναι σε θέση να κατασκευάζει ένα μοναδικό συντακτικό δέντρο για κάθε δυνατή συμβολοσειρά. Για να επιτευχθεί αυτό πρέπει να μην υπάρχουν δύο κανόνες με την ίδια ακολουθία συμβόλων (**reduce–reduce conflict**). Επίσης πρέπει να καθοριστεί πότε ο Bison επιλέγει να κάνει shift και πότε reduce σε περιπτώσεις που τα σύμβολα της στοίβας ικανοποιούν έναν κανόνα αλλά -**μιας και ο Bison είναι LR(1)**- αν ληφθεί υπόψιν και το επόμενο (lookahead) σύμβολο μπορεί να οδηγήσει στην ικανοποίηση κι ενός άλλου κανόνα ταυτόχρονα (**shift–reduce conflict**).

Για την εξάλειψη “παθολογικών” φαινομένων όπως τα παραπάνω που οδηγούν στη δημιουργία **περισσότερων του ενός συντακτικών δέντρων** λαμβάνεται υπόψιν η προτεραιότητα των συμβόλων, η οποία καθορίζεται από την σειρά δήλωσης των συμβόλων μέσα στον κώδικα του Bison (προτεραιότητα από κάτω προς τα πάνω, αντίθετη από αυτή το Flex). Η προτεραιότητα λαμβάνεται υπόψιν και στην περίπτωση των αριθμητικών πράξεων, η σειρά δήλωσης των οποίων έγινε όπως ακριβώς ζητήθηκε στην εκφώνηση.

Αρχικά να σημειωθεί πως μέσα στον κώδικά του συντακτικού αναλυτή που αναπτύχθηκε χρησιμοποιήθηκαν οι παρακάτω εντολές οι οποίες εμφανίζουν μηνύματα λάθους κατά τη χρήση του εκτελέσιμου compiler που σχετίζονται με τυχόν συντακτικά λάθη που μπορεί να έχει το πρόγραμμα που βρίσκεται στο στάδιο της μεταγλώττισης από τον FC_compiler.

%define parse.trace

%define parse.error verbose

Definitions Section

Σε αυτό το κομμάτι του συντακτικού αναλυτή ορίσθηκαν όλα τα **τερματικά σύμβολα** της φανταστικής γλώσσας FC, τόσο αυτά που μας δόθηκαν από την εκφώνηση της άσκησης όσο και όλα αυτά που **επινοήθηκαν** κατά την ανάπτυξη του συντακτικού αναλυτή για την κάλυψη των αναγκών της περιγραφής των συντακτικών κανόνων. Ταυτόχρονα με τον παραπάνω ορισμό τους, όπου χρειαζόταν προσδιορίστηκε επίσης και ο αντίστοιχος $τύπος$[1] των συμβόλων (επειδή θα χρειαστεί να εμφανίσουμε το περιεχόμενό τους) καθώς επίσης και η προσεταιριστικότητά τους όπως ακριβώς δόθηκε στην εκφώνηση της άσκησης.

Π.χ.

```
/* Terminal symbols */
%token KW_STATIC;
%token KW_BOOLEAN;
%token KW_INTEGER;
```

Τύπος:

Π.χ.

```
%token <str> IDENT
%token <intNum> POSINT
%token <doubleNum> REAL
```

1. Βλέπε union παραπάνω.

Είναι προφανές πως σε όλα τα τερματικά σύμβολα που επινοήθηκαν δόθηκε ως τύπος <str>, τύπου string (char*) δηλαδή, καθώς όλα θα χρειαστεί να εμφανιστούν κατά την χρήση του συντακτικού αναλυτή.

Π.χ.

```
/* Variables terminal symbols */
%type <str> programs_body
%type <str> variables
%type <str> data_type
```

Προσεταιριστικότητα:

Π.χ.

```
/* Set left/right associativity */
%left KW_OR OP_OR
%left KW_AND OP_AND
```

Δύο περιπτώσεις που αξίζει να αναφερθούν είναι πως χειρίζεται ο parser την περίπτωση **if – if .. else** καθώς την διαφορά της προσεταιριστικότητας μεταξύ των **KW_PLUS και KW_MINUS** όταν αυτά χρησιμοποιούνται μία ως πρόσημα (αριστερή προσεταιριστικότητα) και μία ως οι αντίστοιχες πράξεις(αριστερή προσεταιριστικότητα).

if – if .. else case

Όταν το else γίνει το lookahead token ο parser μπορεία να κάνει και shift (if .. else κανόνας) και reduce (if κανόνας) (shift–reduce conflict). Για να λυθεί λοιπόν αυτό το shift–reduce δημιουργείται έναν λανθάνον κανόνας (**%precedence IF_PRECEDENCE**) ο οποίος τοποθετείται στο τέλος του κανόνα if και δίνεται μεγαλύτερη προτεραιότητα στο τερματικό token KW_ELSE από αυτό. Η μεγαλύτερη προτεραιότητα στο token KW_ELSE καθορίζεται βάζοντας το "**%precedence IF_PRECEDENCE**" πιο πάνω στο πρόγραμμα από το "**%precedence KW_ELSE**" καθώς έτσι καθορίζεται η προτεραιότητα των tokes στον Bison (από κάτω προς τα πάνω) όπως προαναφέρθηκε.

Έτσι ο parser επιλέγει να κάνει shift αν βρει το KW_ELSE και με αυτόν το τρόπο γλιτώνουμε το Shift/Reduce conflict.

/* Precedence to get rid of the annoying Shift/Reduce */

%precedence IF_PRECEDENCE

%precedence KW_ELSE

KW_IF DEL_LEFT_PARENTHESES expression DEL_RIGHT_PARENTHESES statement %prec IF_PRECEDENCE
KW_IF DEL_LEFT_PARENTHESES expression DEL_RIGHT_PARENTHESES statement KW_ELSE statement

OP_PLUS & OP_MINUS ambiguous precedence

Εντελώς αντίστοιχη λογική χρησιμοποιήθηκε και για την αντιμετώπιση του προαναφερθέντος προβλήματος της αμφιλεγόμενης προσεταιριστικότητας των τελεστών "+" και "-".

%right OP_PLUS OP_MINUS

%left MINUS_PLUS_PRECEDENCE

| OP_PLUS expression %prec MINUS_PLUS_PRECEDENCE

| OP_MINUS expression %prec MINUS_PLUS_PRECEDENCE

Παρατίθεται και το αντίστοιχο section του manual του Bison για περαιτέρω κατανόηση αν κάποιος επιθυμεί να ασχοληθεί και να κατανοήσει την παραπάνω τεχνική εις βάθος.

Bison's Manual link

http://www.gnu.org/software/bison/manual/html_node/Contextual-Precedence.html

Rules Section

Το κομμάτι αυτό περιέχει όλους τους συντακτικούς κανόνες της γλώσσας FC και κατ' επέκταση όλους τους γραμματικούς κανόνες του συντακτικού αναλυτή καθώς και του αντίστοιχου αυτομάτου στοίβας που υλοποιεί ο Bison.

Εντολή template

```
{ $$ = template("%s\n\n%s", $1, $2); }
```

Μια αντίστοιχη εντολή template υπάρχει δεξιά από κάθε κανόνα και η χρησιμότητά της εκεί είναι πως τα περιεχόμενα της (template) θα μεταφερθούν αυτούσια στο αρχείο **.c που θα παραχθεί ύστερα από μια επιτυχημένη μεταγλώττιση ενός προγράμματος **.FC.

Epilogue Section

Σε αυτό το κομμάτι του προγράμματος δηλώθηκε απλά η συνάρτηση Main του συντακτικού αναλυτή η οποία καλεί τη συνάρτηση yyparse() και επιστρέφει ότι της επιστρέψει εκείνη. Η yyparse() με τη σειρά της καλεί επανειλημμένα την yylex() η οποία είναι ουσιαστικά ο λεκτικός αναλυτής FC_lex.

Επιπλέον στοιχεία για τον λεκτικό αναλυτή

Οι συντακτικοί κανόνες είναι χωρισμένοι σε sections μεταξύ τους, τα οποία περιέχουν κανόνες που περιγράφουν σχετικά μεταξύ τους κομμάτια της συντακτικής ανάλυσης, για να είναι πιο εύκολη η ανάγνωση και κατανόηση του προγράμματος.

Compilation προγραμμάτων (Makefile και script)

Όσον αφορά το compilation των προγραμμάτων, γράφτηκε (from scratch) γι' αυτά ένα Makefile κι ένα script, τα Makefile και FC_script αντίστοιχα. Και τα δύο μαζί συνεργάζονται και το τελικό προϊόν της συνεργασίας τους είναι όλη η διαδικασία του compilation και της εξέτασης όλων των παραδειγμάτων που υπάρχουν στον φάκελο Samples (GoodExamples και BadExamples).

Script

Πιο συγκεκιμένα μέσα στο script, στην αρχή συγκεκριμένα, δηλώνονται τα directories των αρχείων και στη συνέχεια καλείται η Makefile ούτως ώστε να διαπιστωθεί αν έχει γίνει κάποιο update σε κάποιο αρχείο κώδικα και κατ' επέκταση αν χρειάζεται να γίνει ξανά ολόκληρη η διαδικασία του compilation από την αρχή ή όχι, προτού εξεταστούν (περάσουν από τον FC_compiler) εκ νέου τα παραδείγματα (**.FC αρχεία). Στη συνέχεια με την χρήση μιας διπλής (εμφολευμένης) δομής επανάληψης For το scrip τρέχει ένα-ένα όλα τα άνωθι παραδείγματα και ανάλογα με την τιμή επιστροφής του FC_compiler για το κάθε πρόγραμμα ξεχωριστά εμφανίζει ένα μήνυμα λάθους ή ορθής μεταγλώττισης του προγράμματος (με κόκκινα ή πράσινα γράμματα αντίστοιχα).

Makefile

Στο Makefile αρχείο που υλοποιήθηκε αρχικά δηλώνονται τα C_FLAGS και οι μεταγλωττιστές. Δόθηκε μεγαλύτερη έμφαση στα debugging possibilities του τελικού εκτελέσιμου (για να είναι πιο εύκολο το debugging προφανώς) και γι' αυτόν ακριβώς το λόγο το πρόγραμμα μεταγλωττίζεται by default με τα debugging flags ενεργοποιημένα και όχι τα αντίστοιχα για optimize κώδικα.

Στη συνέχεια ορίζονται τα dependencies όλων των απαιτούμενων προγραμμάτων που απαιτούν compile και δίνονται οι αντίστοιχοι κανόνες. Να σημειωθεί σε αυτό το σημείο ότι δημιουργήθηκαν κανόνες και για το απαιτούμενο .depend αρχείο που περιέχει τα dependencies σε libraries (header files, **.h αρχεία) καθώς και για τον καθαρισμό των παραγόμενων από τα προαναφερθέντα compile προγραμμάτων (**.c και **.o αρχείων).

Το project ολοκληρώθηκε και λειτουργεί άψογα έχοντας:

- 0 Shift/Reduce
- 0 Reduce/Reduce