Βελτιώσεις και εξηγήσεις πάνω στον κώδικα Τμήμα Β3

Διονύσης Νικολόπουλος

7 Ιουνίου 2021

Περιεχόμενα

Ομάδα 15

Αναλυτικά τα μέλη:

Διονύσης Νικολόπουλος AM : 18390126

Θανάσης Αναγνωστόπουλος AM : 18390043

Αριστείδης Αναγνωστόπουλος AM : 16124

Σπυρίδων Φλώρος AM : 141084

Αναλυτικά οι ρόλοι:

Γενικός Συντονιστής: Διονύσης Νικολόπουλος

Υπεύθυνος Τμήματος Εργασίας Β3: Διονύσης Νικολόπουλος

Github Repo

https://github.com/dnnis/uni-C-Analyser

Usernames στο Github

Διονύσης Νικολόπουλος : dnnis

Θανάσης Αναγνωστόπουλος : ThanasisAnagno

Αριστείδης Αναγνωστόπουλος : Aris-Anag

Σπυρίδων Φλώρος : spirosfl

Η εργασία αυτή πραγματοποιήθηκε με χρήση LaTeX

1 Εισαγωγή

Σε αυτό το έγγραφο θα σας ενημερώσω για τις αλλαγές πάνω στον κώδικά μας μέχρι το μέρος B3.
Θα εξηγηθεί ο κώδικας τμηματικά, θα δωθούν πρόσθετες εξηγήσεις, μαζί με αυτές που δίνονται απο τα σχόλια στον κώδικα τον ίδιο.

1.1 Σημειώσεις για τους φακέλους

- **bison-part**: Σε αυτόν τον κατάλογο βρίσκονται τα αρχεία πηγαίου κώδικα του bison.
- **flex-part**: Σε αυτόν τον κατάλογο βρίσκονται τα αρχεία πηγαίου κώδικα του flex.
- **compile-room**: Σε αυτόν τον κατάλογο βρίσκεται το makefile, το οποίο αντιγράφει τα απαραίτητα αρχεία από τους προαναφερόμενους δύο καταλόγους στον τρέχοντα (compile-room), και με αυτά τα αρχεία κάνει compile στο τελικό μας πρόγραμμα, που ονομάζεται uni-c-analyser.

2 Τμηματικά ο κώδικας - Εξηγήσεις

2.1 FLEX

2.1.1 Includes και μεταβλητές

```
/* Kwdikas C gia orismo twn apaitoumenwn header files kai twn
   metablhtwn.
   Otidhpote anamesa sta %{ kai %} metaferetai autousio sto arxeio C
   pou
   tha dhmiourghsei to Flex. */

%{

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "bison-SA.tab.h"

// Για να μετράμε τις κολώνες
int columns = 1;
// Αρχικοποιούμε τις μεταβλητές για το άθροισμα των σωστών και λάθος
   λέξεων
int cor_words = 0;
int inc_words = 0;
// Για να καταφέρνουμε να δίνουμε στον χρήστη σωστό output.
char panic_cause_char[100];

%}
```

Εδώ βλέπουμε τα include μας, που είναι απαραίτητα τόσο για την λειτουργεία του προγράμματος (πχ. '#include <stdlib>' κτλ) όσο και για την σύνδεση του flex με του bison (πχ. '#include "bison-SA.tab.h"')
Επίσης, βλέπουμε τις μεταβλητές που ορίζουμε για την ομαλή διεπαφή του χρήστη με το πρόγραμμα.
Πλέον το πρόγραμμά μας μετρά κολώνες και μπορεί να κατευθύνει τον χρήστη στο ακριβές σημείο που το πρόβλημα δημιουργήθηκε (με κάποιο άγνωστο token).
Επίσης, σε σύγκριση με το μέρος B2, τώρα το πρόγραμμα μετρά σωστά

τις σωστές και λάθος εκφράσεις.
Τις λάθος λέξεις τις μετρά ο λεκτικός αναλυτής, ο FLEX. Τις λάθος εκφράσεις ο συντατικός αναλυτής, το BISON.
Επίσης, στην μεταβλητή panic_cause_char αποθηκεύουμε την άγνωστη λέξη, και την αναφέρουμε στον χρήστη μετέπειτα.

2.1.2 TOKENS, Κανονικές Εκφράσεις, Καταστάσεις

```
/* Onomata kai antistoixoi orismoi (ypo morfh kanonikhs ekfrashs).
   Meta apo auto, mporei na ginei xrhsh twn onomatwn (aristera) anti
    twn,
   synhthws idiaiterws makroskelwn kai dysnohtwn, kanonikwn ekfrasewn
    */

SEMI                ;
HASH                #
TILDE               ~
NEQ                 !=
MOD                 \%
POW                 \^
DOT                 \.
COMMA               \,
COLON               \:
AMPER               \&
PAR_END             \)
PAR_START           \(
BRACE_END           \}
BRACE_START         \{
BRACKET_END         \]
BRACKET_START       \[
LOGICAL_OR          \|\|
TYPE_EQ             ==?
TYPE_DIV            \/=?
TYPE_MULTI          \*=?
TYPE_EXCLA          \!=?
TYPE_AMPER          \&\&
TYPE_LESSER         \<=?
TYPE_GREATER        \>=?
WHITESPACE          [ \t]
TYPE_PLUS           \+[\+=]?
TYPE_MINUS          \-[\-=]?
STRING              '.*'|\".*\"
```

```
33 INTCONST             0|[1-9]+[0-9]*
34 COMMENT              \/\*(.|\n)*?\*\/|\/\/.*
35 IDENTIFIER           [a-zA-Z_]([0-9_a-zA-Z]*)
36 FLOAT                [0-9]+\.[0-9]+|[0-9]+\.[0-9]+e[0-9]+
```

Βλέπουμε τις κανονικές εκφράσεις με τις οποίες ο λεκτικός αναλυτής μας ανιχνεύει τις λέξεις του αναλυόμενου κώδικα, και τις συσχετίζει με ένα *μοναδικό* token.
Στο τέλος έχουμε και τους ορισμούς για τις 3 καταστάσεις τις οποίες ορίσαμε για να λειτουργεί ορθά ο λεκτικός αναλυτής.

```
1 %x REALLYEND
2 %x PREPANIC
3 %x PANIC
```

Αυτές είναι:

- **Κατάσταση REALLYEND** είναι η κατάσταση στην οποία ο λεκτικός αναλυτής έχει ήδη απο τον συντακτικό αναλυτής το μύνημα ότι ο πρώτος έχει λάβει τις τελευταίες δράσεις πριν τον τερματισμό του προγράμματος, και αρχίζει να κλείνει "πραγματικά" το πρόγραμμα. (Πριν την κατάσταση αυτή στέλνει μήνυμα EOF στον BISON, που παίρνει δράσεις που θα αναλύσουμε παρακάτω. Μετά από τις δράσεις αυτές, ο FLEX μπαίνει στην κατάσταση REALLYEND)

- **Κατάσταση PANIC** είναι η κατάσταση στην οποία ο λεκτικός αναλυτής έχει συναντίσει ένα άγνωστο token (UNKNOWN TOKEN) και προσπαθεί να κάνει επανάκτιση κανονικής λειτουργίας.

- **Κατάσταση PREPANIC** είναι η κατάσταση στην οποία ο λεξικός μας αναλυτής ειδοποιεί τον συντακτικό αναλυτή ότι πρόκειται να βρεθει (ο δεύτερος) σε κατάσταση πανικού.

2.1.3 Κώδικας κατά την ανίχνευση λεκτικών μονάδων

```
%%
{MOD}           {cor_words++; columns++; return MOD;}
{POW}           {cor_words++; columns++; return POW;}
{DOT}           {cor_words++; columns++; return DOT;}
{SEMI}          {cor_words++; columns++; return SEMI;}
{HASH}          {cor_words++; columns++; return HASH;}
{COMMA}         {cor_words++; columns++; return COMMA;}
{PAR_END}       {cor_words++; columns++; return PAR_END;}
{PAR_START}     {cor_words++; columns++; return PAR_START;}
{BRACE_END}     {cor_words++; columns++; return BRACE_END;}
{LOGICAL_OR}    {cor_words++; columns++; return LOGICAL_OR;}
{BRACE_START}   {cor_words++; columns++; return BRACE_START;}
{BRACKET_END}   {cor_words++; columns++; return BRACKET_END;}
{BRACKET_START} {cor_words++; columns++; return BRACKET_START;}
```

Συνεχίζοντας, παρατηρούμε πώς αυξάνουμε τον αριθμό των σωστών λέξεων για κάθε λέξη που ανιχνεύεται σωστά, αυξάνοντας και το αριθμό των κολωνών κατάλληλα επίσης.
Συνεχίζοντας σε λεκτικές μονάδες με περισσότερα γράμματα απο 1:

```
{FLOAT}    {cor_words++; columns += strlen(yytext); return FLOAT;}
{STRING}   {cor_words++; columns += strlen(yytext); return STRING;}
{INTCONST} {cor_words++; columns += strlen(yytext); return INTCONST;}
```

Εδώ είναι ξεκάθαρο ότι χρησιμοποιούμε την συνάρτηση strlen η οποία μετρά τις λεκτικές μονάδες "απρόβλεπτου" μήκους και προσθέτει τον αριθμό γραμμάτων τους στην μεταβλητή μέτρησης κολωνών columns. Ακολουθούν τα ονόματα και τα keywords.

```
{IDENTIFIER}    {
if      ( !strcmp(yytext,"do"      ))
  {cor_words++; columns += strlen(yytext); return KEYWORD;}
else if ( !strcmp(yytext,"while"   ))
  {cor_words++; columns += strlen(yytext); return KEYWORD;}
else if ( !strcmp(yytext,"break"   ))
  {cor_words++; columns += strlen(yytext); return KEYWORD;}
else if ( !strcmp(yytext,"if"      ))
  {cor_words++; columns += strlen(yytext); return KEYWORD_IF;}
else if ( !strcmp(yytext,"struct"  ))
  {cor_words++; columns += strlen(yytext); return KEYWORD_STR;}
else if ( !strcmp(yytext,"for"     ))
```

```
  {cor_words++; columns += strlen(yytext); return KEYWORD_FOR;}
else if ( !strcmp(yytext,"return"  ))
  {cor_words++; columns += strlen(yytext); return KEYWORD_RET;}
else if ( !strcmp(yytext,"case"    ))
  {cor_words++; columns += strlen(yytext); return KEYWORD_CASE;}
else if ( !strcmp(yytext,"else"    ))
  {cor_words++; columns += strlen(yytext); return KEYWORD_ELSE;}
else if ( !strcmp(yytext,"func"    ))
  {cor_words++; columns += strlen(yytext); return KEYWORD_FUNC;}
else if ( !strcmp(yytext,"void"    ))
  {cor_words++; columns += strlen(yytext); return KEYWORD_VOID;}
else if ( !strcmp(yytext,"sizeof"  ))
  {cor_words++; columns += strlen(yytext); return KEYWORD_SIZE;}
else if ( !strcmp(yytext,"include" ))
  {cor_words++; columns += strlen(yytext); return KEYWORD_INCL;}
else if ( !strcmp(yytext,"continue"))
  {cor_words++; columns += strlen(yytext); return KEYWORD_CONT;}
else if ( !strcmp(yytext,"switch"  ))
  {cor_words++; columns += strlen(yytext); return KEYWORD_SWITCH;}
else if ( !strcmp(yytext,"int"     ))
  {cor_words++; columns += strlen(yytext); return KEYWORD_VAR_TYPE;}
else if ( !strcmp(yytext,"char"    ))
  {cor_words++; columns += strlen(yytext); return KEYWORD_VAR_TYPE;}
else if ( !strcmp(yytext,"long"    ))
  {cor_words++; columns += strlen(yytext); return KEYWORD_VAR_TYPE;}
else if ( !strcmp(yytext,"short"   ))
  {cor_words++; columns += strlen(yytext); return KEYWORD_VAR_TYPE;}
else if ( !strcmp(yytext,"float"   ))
  {cor_words++; columns += strlen(yytext); return KEYWORD_VAR_TYPE;}
else if ( !strcmp(yytext,"const"   ))
  {cor_words++; columns += strlen(yytext); return KEYWORD_VAR_TYPE;}
else if ( !strcmp(yytext,"double"  ))
  {cor_words++; columns += strlen(yytext); return KEYWORD_VAR_TYPE;}
else
  {cor_words++; columns += strlen(yytext); return IDENTIFIER;}
}
```

Εδώ έχουμε το ενδεχόμενο κάποια έγκυρη λεκτική μονάδα που ανιχνεύσαμε να είναι *δευσμευμένη λέξη* της Uni-C.
Αν είναι, επιστρέφουμε το κατάλληλο token.
Αν όχι τότε επιστρέφουμε απλά token IDENTIFIER.

Πηγαίνοντας σε πιο πολύπλοκες λεκτικές μονάδες όπως οι τελεστές:

```
{TYPE_EXCLA}    { if (!strcmp(yytext, "!=")) {cor_words++; columns +=
    2; return NEQ;        } else    { columns++; return EXCLA;  }}
{TYPE_EQ}       { if (!strcmp(yytext, "==")) {cor_words++; columns +=
    2; return EQQ;        } else    { columns++; return EQ;     }}
{TYPE_DIV}      { if (!strcmp(yytext, "/=")) {cor_words++; columns +=
    2; return EQ_DIV;     } else    { columns++; return DIV;    }}
{TYPE_MULTI}    { if (!strcmp(yytext, "*=")) {cor_words++; columns +=
    2; return EQ_MULTI;   } else    { columns++; return MULTI;  }}
{TYPE_LESSER}   { if (!strcmp(yytext, "<=")) {cor_words++; columns +=
    2; return LESSER_EQ;  } else    { columns++; return LESSER; }}
{TYPE_GREATER}  { if (!strcmp(yytext, ">=")) {cor_words++; columns +=
    2; return GREATER_EQ; } else    { columns++; return GREATER;}}
{TYPE_AMPER}    { if (!strcmp(yytext, "&&")) {cor_words++; columns +=
    2; return LOGICAL_AND;} else    { columns++; return AMPER;  }}
{TYPE_MINUS}    { if (!strcmp(yytext, "--")) {cor_words++; columns +=
    2; return MINUSMINUS; } else if (!strcmp(yytext, "-=")) { columns
    +=2; return EQ_MINUS; } else { columns++; return MINUS;}}
{TYPE_PLUS}     { if (!strcmp(yytext, "++")) {cor_words++; columns +=
    2; return PLUSPLUS;   } else if (!strcmp(yytext, "+=")) { columns
    +=2; return EQ_PLUS;  } else { columns++; return PLUS; }}
```

Έχουμε ιδιαίτερη διαχείρηση των τελεστών για να είμαστε σίγουροι ότι ο λεκτικός μας αναλυτής δεν "μπερδεύει" για παράδειγμα, τον τελεστή "++" από τον "+", καθώς συντατικά είναι πολύ διαφορετική η συμπεριφορά τους.
Στην συνέχεια δίνονται οδηγίες στον λεκτικό αναλυτή για την διαχείρηση του κενού, της κανούργιας γραμμής και των σχολίων του κώδικα.

```
{WHITESPACE} { columns++; }
{COMMENT}    { /*Do nothing, comment*/ }
\n           { columns=1; /*Start from zero cols again*/ return NEWLINE; }
```

Για τα κενά ο λεκτικός αναλυτής απλά αυξάνει τις κολώνες, για τα σχόλια δεν κάνει τίποτα, ενώ για τις καινούργιες γραμμές επαναφέρει την μεταβλητή μέτρησης κολωνών columns στην αρχική τιμή 1.
Τελειώνοντας απο τον κώδικα του λεκτικού αναλυτή, βλέπουμε το πιο πολύπλοκο κομμάτι του.

2.1.4 Καταστάσεις (Πανικού!)

Μπαίνοντας στο πιό πολύπλοκο κομμάτι του λεκτικού μας αναλυτή, έχουμε την διαχείρηση λανθασμένων λέξεων, οι οποίες ρίχνουν τον αναλυτή μας σε μία κατάσταση "πανικού".
Απο αυτή την κατάσταση προσπαθεί μετά να "ξεφύγει" με το να αναφέρει το λάθος στον συντακτικό αναλυτή και να επανέλθει στην αρχική κατάσταση <INITIAL>, επανακκινόντας την λεκτική ανάλυση του υπόλοιπου αρχείου.

```
/*Εδώ το flex πιάνει"" οποιονδήποτε άλλο χαρακτήρα που
 δεν περιγράφεται απο τις παραπάνω κανονικές εκφράσεις.*/

<INITIAL>.  { BEGIN(PREPANIC); strcpy(panic_cause_char,yytext);
    inc_words++; return UNKNOWN;}
<PREPANIC>. { BEGIN(PANIC); printf("Column: %d Unknown word: '%s%s",
    columns,panic_cause_char,yytext);}
<PANIC>{WHITESPACE} { columns++; printf("'\n"); BEGIN(INITIAL);}
<PANIC>\n           { columns=1; printf("'\n"); BEGIN(INITIAL);}
<PANIC>.            { ECHO; }

/*Εδώ καλούμε ένα τμήμα κώδικα που μας βοηθά να δώσουμε ένα token
  στον bison για να δηλώσουμε το τέλος του αρχείου, αποτρέποντας
  όμως τον bison να τερματίζει άμεσα την εκτέλεση. Έτσι,
  καταφέρνουμε να εκτελούμε την συνάρτηση print_report() στο
  bison-SA.y, για να ανεφέρουμε τον αριθμό των σωστών και
  λανθασμένων εκφράσεων. Έπειτα, αναφέρουμε με τον λεκτικό
  αναλυτή τις λάθος λέξεις*/

<INITIAL><<EOF>>  { BEGIN(REALLYEND);
                    printf("*---- RUN REPORT: ---------------------*\n"
                           "|- Words:\n"
                           "| Number of  correct  words       : %d\n"
                           "| Number of incorrect words       : %d\n"
                           "*-------------------------------------*\n"
                           ,cor_words, inc_words);
                  return EOP; }
 /*Εδώ, μετά την πάροδο των προηγούμενων, πραγματικά""
  τερματίζουμε την εκτέλεση του flex, έχουμε ήδη τυπώσει την αναφορά
  με την print_report() με το bison, και αρχίζουμε να τερματίζουμε
  το πρόγραμμα συνολικά.*/
<REALLYEND><<EOF>> {yyterminate();}
%%

```

```
/* Το πρόγραμμα αυτό δεν έχει main(), καθώς δεν τρέχει αυτόνομα,είναι
 απλά ο λεκτικός αναλυτής, η συντακτική ανάλυση γίνεταιαπό
 τον bison. */
```

2.2 BISON

2.2.1 Αρχικοποίηση συναρτήσεων και μεταβλητών

```
%{
/* Orismoi kai dhlwseis glwssas C. Otidhpote exei na kanei me orismo h
   arxikopoihsh metablhtwn & synarthsewn, arxeia header kai dhlwseis #
    define
   mpainei se auto to shmeio */
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
extern int yylex(void);
extern int yyparse(void);
void yyerror(char *);
void print_report(int,int);
void print_valid (char *);
// Αρχικοποιούμε τον pointer για τη εισαγωγή δεδομένων με αρχείο και
    όχι απο το
// stdin
extern FILE *yyin;
// Αρχικοποιούμε τις μεταβλητές για το άθροισμα των σωστών και λάθος
    εκφράσεων
int cor_expr  = 0;
int inc_expr  = 0;
// Για την γραμμή που αρχίζει μία συνάρτηση
int brack_start_line=0;

// Για να αναφέρουμε απο που ως που μια συνάρτηση αρχίζει.
int function_start_line=0;
int function_started_flag=0;

// Για την μέτρηση γραμμών
int line=1;
%}
```

Αρχίζουμε τον κώδικα του συντακτικού μας αναλυτή έτσι όπως αρχίσαμε και τον λεκτικό. Με τα includes βιβλιοθηκών που είναι αναγκάια στην λειτουργία του κώδικα.
Εδώ έχουμε ωστόσο και κάποιες διαφορές με το αντίστοιχο τμήμα του λεκτικού αναλυτή.
Αρχικά, έχουμε την αρχικοποίηση στον κώδικά μας των εξωτερικών συναρτήσεων yylex και yyparse, που υλοποιούνται από τον κώδικά μας στον λεκτικό αναλυτή. Αυτό είναι αναγκαίο για την σύνδεση και συνεργασία του FLEX με του BISON (Βλέπε παράγραφο 2.1.1).
Μετά, έχουμε τις λοιπές μεταβλητές που εξηγούνται μέσω των σχολίων πάνω στον ίδιο τον κωδικα.

2.2.2 Αρχικοποίηση TOKENS

Συνεχίζοντας, αρχικοποιούμε όλα τα TOKEN *τα οποία λαμβανουμε* απο τον λεκτικό αναλυτη.
Για παράδειγμα, απουσιάζει το TOKEN "WHITESPACE", που ο λεκτικός αναλυτής ποτέ δεν επιστρέφει στο συντακτικό αναλυτη.

```
%union
{
    int    ival;
    char*  sval;
    float  fval;
    double dval;
}

// Ορισμός των λεκτικών μονάδων
%token EOP
    UNKNOWN
    <sval> DOT
    <sval> SEMI
    <sval> HASH
    <sval> COLON
    <sval> COMMA
    <sval> FLOAT
    <dval> DOUBLE
```

```
19      <fval> STRING
20      <sval> NEWLINE
21      <sval> KEYWORD
22      <ival> INTCONST
23      <sval> IDENTIFIER
24      <sval> KEYWORD_IF
25      <sval> AMPER EXCLA
26      <sval> KEYWORD_RET
27      <sval> KEYWORD_FOR
28      <sval> KEYWORD_STR
29      <sval> KEYWORD_ELSE
30      <sval> KEYWORD_SIZE
31      <sval> KEYWORD_CONT
32      <sval> KEYWORD_CASE
33      <sval> KEYWORD_INCL
34      <sval> KEYWORD_FUNC
35      <sval> KEYWORD_VOID
36      <sval> KEYWORD_SWITCH
37      <sval> KEYWORD_VAR_TYPE
38      <sval> PAR_START PAR_END
39      <sval> BRACE_START BRACE_END
40      <sval> LOGICAL_OR LOGICAL_AND
41      <sval> BRACKET_START BRACKET_END
42      <sval> GREATER LESSER GREATER_EQ LESSER_EQ
43      <sval> EQQ EQ NEQ EQ_MULTI EQ_DIV EQ_PLUS EQ_MINUS
44      <sval> PLUS PLUSPLUS MINUS MINUSMINUS DIV MOD MULTI POW
45
46  // Ορισμός προτεραιώτητας στα tokens
47  %left  POW
48  %left  PLUS MINUS
49  %left  DIV MULTI
```

2.2.3 Γραμματικοί κανόνες

Οι κανόνες με τους οποίους ο συντακτικός μας αναλυτής ελέγχει αν τα συνεχή token που δέχεται από τον λεκτικό αναλυτή διαμορφώνουν σωστά μια πρόταση στην Uni-C ή όχι.
Ο BISON δέχεται τους κανόνες αυτούς γραμμένους σε μορφή EBNF. (Extended Backus-Naur Form)

```
1  %%
2  /* Orismos twn grammatikwn kanonwn. Kathe fora pou antistoixizetai enas
```

```
3     grammatikos kanonas me ta dedomena eisodou, ekteleitai o kwdikas C
      pou
4     brisketai anamesa sta agkistra. H anamenomenh syntaksh einai: onoma
      :
5     kanonas { kwdikas C } */
6  program:
7       program valid
8       |
9       ;
10
11
12 /* Εδώ ορίζεται το τι μπορεί να είναι κομμάτι μίας έκφρασης.
13    Ένας χαρακτήρας ή ένας αριθμός */
14 expr_part:
15        FLOAT
16      | STRING
17      | DOUBLE
18      | KEYWORD
19      | INTCONST
20      | IDENTIFIER
21      | UNKNOWN { printf("X\tLine:  %d \t",line); }
22      ;
23
24 // Εδώ ορίζονται οι τελεστές
25 operator:
26        EQ
27      | EQQ
28      | NEQ
29      | DIV
30      | POW
31      | PLUS
32      | MINUS
33      | MULTI
34      | EQ_DIV
35      | EQ_PLUS
36      | EQ_MULTI
37      | EQ_MINUS
38      ;
39
40 in_de_crement_operator:
41      | MINUSMINUS
42      | PLUSPLUS
43      ;
44
```

```
// Εδώ ορίζονται ποιές είναι οι εκφράσεις υπο επεξεργασία
expr_proc:
      expr_part operator expr_part EQ expr_part
    | expr_part operator expr_part
    | expr_part in_de_crement_operator
    | in_de_crement_operator expr_part
    ;

/* Εδώ ορίζεται το σώμα"" του κώδικα, δηλαδή ένας αριθμός συντακτικά
   σωστών εκφράσεων. */
body:
    body valid
    | valid
    |
    ;

elements:
    expr_part COMMA elements
    | expr_part
    ;

// Εδώ ορίζεται τι μπορεί να βρίσκεται μέσα σε αγγύλες
in_brack:
    BRACKET_START elements BRACKET_END

// Εδώ ορίζεται τι μπορεί να βρίσκεται μέσα σε άγκυστρο
in_brace:
    BRACE_START body BRACE_END

struct:
      KEYWORD_STR IDENTIFIER in_brace
    | KEYWORD_STR IDENTIFIER NEWLINE in_brace
    ;

loops:
    for_grammar

// Εδώ ορίζεται τι μπορεί να είναι ορίσματα μιας συνάρτησης
arguments:
      arguments expr_part COMMA expr_part
    | expr_part COMMA expr_part
    | KEYWORD_VOID
    |
    ;
```

```
89
90  // Εδώ ορίζεται τι θεωρείται ορισμός μιας συνάρτησης
91  func_par:
92        KEYWORD_FUNC IDENTIFIER PAR_START arguments PAR_END {cor_expr++;
        print_valid("arguments"); }
93      | KEYWORD_FUNC IDENTIFIER PAR_START expr_part PAR_END {cor_expr++;
        print_valid("argument"); }
94      ;
95
96  // Εδώ ορίζεται τι θεωρείται ορισμός μιας μεταβλητής
97  declaration:
98        KEYWORD_VAR_TYPE IDENTIFIER
99      | KEYWORD_VAR_TYPE IDENTIFIER EQ expr_proc
100     | KEYWORD_VAR_TYPE IDENTIFIER in_brack EQ expr_proc
101     | KEYWORD_VAR_TYPE IDENTIFIER in_brack EQ BRACE_START elements
        BRACE_END
102     | KEYWORD_VAR_TYPE IDENTIFIER in_brack
103     | KEYWORD_VAR_TYPE IDENTIFIER EQ sizeof
104     ;
105
106 // Εδώ ορίζεται τι θεωρείται ανάθεση σε μεταβλητή
107 assignment:
108     IDENTIFIER EQ expr_proc
109
110 // Ο κανόνας για τις επιστροφές
111 return:
112     KEYWORD_RET expr_proc
113     | KEYWORD_RET expr_part
114     ;
115
116 // Ο κανόνας για τα includes
117 include:
118     HASH KEYWORD_INCL LESSER IDENTIFIER DOT IDENTIFIER GREATER
119     | HASH KEYWORD_INCL STRING
120     ;
121
122 cases:
123     KEYWORD_CASE COLON valid NEWLINE cases
124     | KEYWORD_CASE COLON valid NEWLINE
125
126 case_grammar:
127     KEYWORD_SWITCH PAR_START expr_proc PAR_END BRACE_START cases
        BRACE_END
```

```
128     | KEYWORD_SWITCH PAR_START expr_part PAR_END BRACE_START cases
    BRACE_END
129     ;
130
131 else_grammar:
132     KEYWORD_ELSE in_brace
133
134 if_grammar:
135       KEYWORD_IF PAR_START expr_proc PAR_END in_brace
136     | KEYWORD_IF PAR_START expr_proc PAR_END expr_proc NEWLINE
137     ;
138
139 for_grammar:
140       KEYWORD_FOR PAR_START for_args PAR_END in_brace
141     | KEYWORD_FOR PAR_START for_args PAR_END expr_proc NEWLINE
142     ;
143
144 for_args:
145       expr_proc SEMI expr_proc SEMI expr_proc
146     | SEMI expr_proc SEMI expr_proc
147     | expr_proc SEMI SEMI expr_proc
148     | expr_proc SEMI SEMI
149     | SEMI expr_proc SEMI
150     | SEMI SEMI expr_proc
151     | SEMI SEMI
152     ;
153
154 // Ο κανόνας αυτός χρησιμοποιήται μαζί με το sizeof πχ(. sizeof(smth)
    * 10)
155 // Με το "* 10" να είναι το "half_expr"
156 half_expr:
157       operator IDENTIFIER
158     | operator INTCONST
159     | operator DOUBLE
160     | operator FLOAT
161     ;
162
163 // Ο κανόνας για το sizeof
164 sizeof:
165       KEYWORD_SIZE PAR_START KEYWORD_VAR_TYPE PAR_END
166     | KEYWORD_SIZE PAR_START KEYWORD_VAR_TYPE PAR_END half_expr
167     ;
168
169 // Εδώ είναι όλοι οι κανόνες των if/else/case
```

```
conditionals:
      if_grammar
    | else_grammar
    | case_grammar
    ;

// Εδώ ορίζεται τι θεωρείται συντακτικά σώστο
valid:
     return      SEMI { cor_expr++; print_valid("return");}
   | sizeof      SEMI { cor_expr++; print_valid("sizeof");}
   | include     SEMI { cor_expr++; print_valid("include");}
   | expr_proc   SEMI { cor_expr++; print_valid("expression");}
   | assignment  SEMI { cor_expr++; print_valid("assignment");}
   | declaration SEMI { cor_expr++; print_valid("declaration");}
   | loops            { cor_expr++; print_valid("loop clause");}
   | in_brace         { cor_expr++;
                        if( function_started_flag)
                        {
                            function_started_flag=0;
                            if (line == function_start_line)
                            {
                               printf("O\tLine:  %d \tValid function
    body!\n",function_start_line);
                            } else if (line >= function_start_line) {
                               printf("O\tLines: %d-%d\tValid function
    body!\n",function_start_line, line);
                            }
                        } else {
                            function_started_flag=1;
                            function_start_line=line;
                        }
                      }
   | struct SEMI      { cor_expr++; print_valid("struct");}
   | func_par         { cor_expr++; print_valid("function declaration")
    ;}
   | conditionals     { cor_expr++; print_valid("conditional clause");
     }
   | NEWLINE          { line++; }
   | EOP              { print_report(cor_expr,inc_expr); }
   | error            { inc_expr++;}
   ;

%%
```

2.2.4 Οι συναρτήσεις μας

Οι ορισμένες απο εμάς συναρτήσεις είναι οι εξής:

```
// Αυτή η συνάρτηση τυπώνει το output του συντακτικού αναλυτη όταν
// μια αποδεκτή έκφραση ανιχνευθεί, με σταθερό format.
void print_valid (char * type) {
    printf("O\tLine:  %d \tValid %s!\n"    ,line, type);
}
// Αυτή η συνάρτηση τυπώνει το πλήθος των σωστών και λάθος λέξεων
   και εκφράσεων
// Ενεργοποιήται μόλις ο bison δεχθεί token EOP
// (End of Parse, δίνεται στο τέλος του αρχείου)
void print_report (int cor_expr,int inc_expr) {
    printf("|- Expressions:\n"
           "| Number of  correct  expressions : %d\n"
           "| Number of incorrect expressions : %d\n"
           "*-------------------------------------*\n"
           ,cor_expr, inc_expr);
}

/* H synarthsh yyerror xrhsimopoieitai gia thn anafora sfalmatwn.
   Sygkekrimena kaleitai
   apo thn yyparse otan yparksei kapoio syntaktiko lathos. Sthn
   parakatw periptwsh h
   synarthsh epi ths ousias typwnei mhnyma lathous sthn othonh. */
void yyerror(char *s) {
    fprintf(stderr, "X\tLine:  %d \tError: %s\n",line, s);
}
```

2.2.5 Η συνάρτηση main

Λόγω του κώδικα ifdef ... endif πριν την main, είναι δυνατό να κάνει compile ο χρήστης το πρόγραμμα έτσι ώστε να δίνει output ο BISON για το πού βρίσκεται το stack και τι tokens δέχεται, σε ποιό κανόνα "κινέιται" την δεδομένη στιγμή. (Βλέπε κεφάλαιο 4)

```
//Αναγκαίες εντολές για να γίνεται το debugging στον Bison
#ifdef YYDEBUG
  yydebug = 1;
#endif

```

```
/* H synarthsh main pou apotelei kai to shmeio ekkinhshs tou programmatos.
   Sthn sygkekrimenh periptwsh apla kalei thn synarthsh yyparse tou Bison
   gia na ksekinhsei h syntaktikh analysh. */
int main(void) {
    // Open a file handle to a particular file:
    FILE *myfile = fopen("input.txt", "r");
    // Make sure it is valid:
    if (!myfile)
    {
      printf("* Error: cannot open the \"input.txt\" file!");
      return -1;
    }
    // Set Flex to read from it instead of defaulting to STDIN:
    printf("\n*---- ANALYSIS: --------------------*\n");
    yyin = myfile;
    yyparse();
    fclose(myfile);
}
```

Κατα τα άλλα έχουμε μια απλή υλοποίηση διαβάσματος από αρχείο στην C, με κάλεσμα της yyparse για να αναλυθεί το αρχείο λεκτικά από τον λεκτικό αναλυτή πρώτα, που με την σειρά του δίνει tokens για την συντακτική ανάλυση του στον συντακτικό αναλυτή.

3 Compilation

Το makefile μας βρήσκεται μέσα στον κατάλογο `compile-room` και, εφόσον μεταβούμε στον κατάλογο αυτό μπορούμε να δώσουμε 3 εντολές:

- **`make`** για το "κανονικό" compilation. Θα αντιγράψει τα απαράιτητα αρχεία στον κατάλογο compile-room και θα κάνει compile χωρίς πρόσθετες επιλογές το πρόγραμμα uni-c-analyser.
- **`make debug`** για το "debug" compilation. Όπως αναφέραμε και στο κεφάλαιο 2.2.5, λόγω ειδικών ρυθμίσεων μπορούμε να

κάνουμε compile το πρόγραμμα έτσι ώστε ο BISON να δίνει πρόσθετες πληροφορίες για το stack και για τα token που δέχεται, όπως και το ποιόν κανόνα "ακολουθεί" κάθε φορά που δέχεται token.

- **make clean** για το "καθάρισμα" του καταλόγου απο τα πρόσθετα αρχεία που δημιουργούνται από τα compilation.

4 Εκτέλεση

Όταν κάνουμε make το πρόγραμμα, αυτό αυτόματα εκτελέιται για πρώτη φορά και βάζει το output σε ένα αρχείο στον τρέχοντα κατάλογο, που λέγεται "output.txt".
Φυσικά, "χειροκίνητα" μπορούμε να τρέξουμε το αρχείο και με την εντολή `./uni-c-analyser` στον κατάλογο που βρίσκεται το εκτελέσιμο αρχείο.

5 Αρχείο input.txt

Τα αρχέια εισαγωγής βρισκονται στον κατάλογο `input-files`. Από έκει το makefile αντιγράφει κάθε φορά που εκτελείται εντολή `make` ή `make debug` το αρχέιο "input.txt", που μετά χρησιμοποιται για την είσοδο στο uni-c-analyser, αντί για το stdin, αρκεί το εκτελέσιμο και το αρχέιο εισαγωγής να είναι τον ίδιο κατάλογο (κάτι που συμβαίνει αν απλά τρεχτεί το makefile από τον κατάλογο `compile-room`).

6 Ενδεικτική εκτέλεση

To input.txt μας είναι αυτό:

```
#include <stdio.h>;
#include <string.h>;
#include "customlib.h";

struct Car
{
    int   numberplate;
    char  brand[20];
    float eng_capacity;
};

func oneline_function (void) { int nothing; }

func main (argc, argv) {
    int metavliti = 1;
    int variable = 0;
    int res  = variable + 1;
    int res2 = variable * 8;
    int res3 = variable + metavliti;
    int res4 = variable - 1;
    int res5 = 5 + 5;
    int res6 = 100/5;
    int res7 = 700-800;
    int res8 = 1 / 1;
    int res9 = 1 ^ 100;
    int res9 = 10 ^ variable;
    int res10[12] = 1;
    int res11 = sizeof(char) * 10;
    int res12 = sizeof(char) + 10;
    int res13 = sizeof(char);

    if ( variable == true ) {
        someflag=1;
    } else {
        someflag=0;
    }

    //Below are some errors/unknown tokens
    $$a56f$$$
    $$a56
```

```
    $$sdfisd
          $rhfa'$\5[fiw
}
```

Το output.txt μας είναι αυτό:

```
*---- ANALYSIS: --------------------*
O Line:  1  Valid include!
O Line:  2  Valid include!
O Line:  3  Valid include!
O Line:  6  Valid declaration!
O Line:  7  Valid declaration!
O Line:  8  Valid declaration!
O Line:  9  Valid struct!
O Line:  11   Valid arguments!
O Line:  11   Valid function declaration!
O Line:  11   Valid declaration!
O Line:  13   Valid arguments!
O Line:  13   Valid function declaration!
O Line:  14   Valid declaration!
O Line:  15   Valid declaration!
O Line:  16   Valid declaration!
O Line:  17   Valid declaration!
O Line:  18   Valid declaration!
O Line:  19   Valid declaration!
O Line:  20   Valid declaration!
O Line:  21   Valid declaration!
O Line:  22   Valid declaration!
O Line:  23   Valid declaration!
O Line:  24   Valid declaration!
O Line:  25   Valid declaration!
O Line:  26   Valid declaration!
O Line:  27   Valid declaration!
O Line:  28   Valid declaration!
O Line:  29   Valid declaration!
O Line:  32   Valid assignment!
O Line:  33   Valid conditional clause!
O Line:  34   Valid assignment!
O Line:  35   Valid conditional clause!
X Line:  38   Column: 5 Unknown word: '$$a56f$$$'
X Line:  38   Column: 5 Unknown word: '$$a56'
X Line:  38   Column: 6 Unknown word: '$$sdfisd'
X Line:  38   Column: 12 Unknown word: '$rhfa'$\5[fiw'
O Lines: 11-38  Valid function body!
*---- RUN REPORT: --------------------*
```

```
|- Words:
| Number of  correct  words       : 159
| Number of incorrect words       : 4
*-------------------------------------*
|- Expressions:
| Number of  correct  expressions : 34
| Number of incorrect expressions : 4
*-------------------------------------*
```