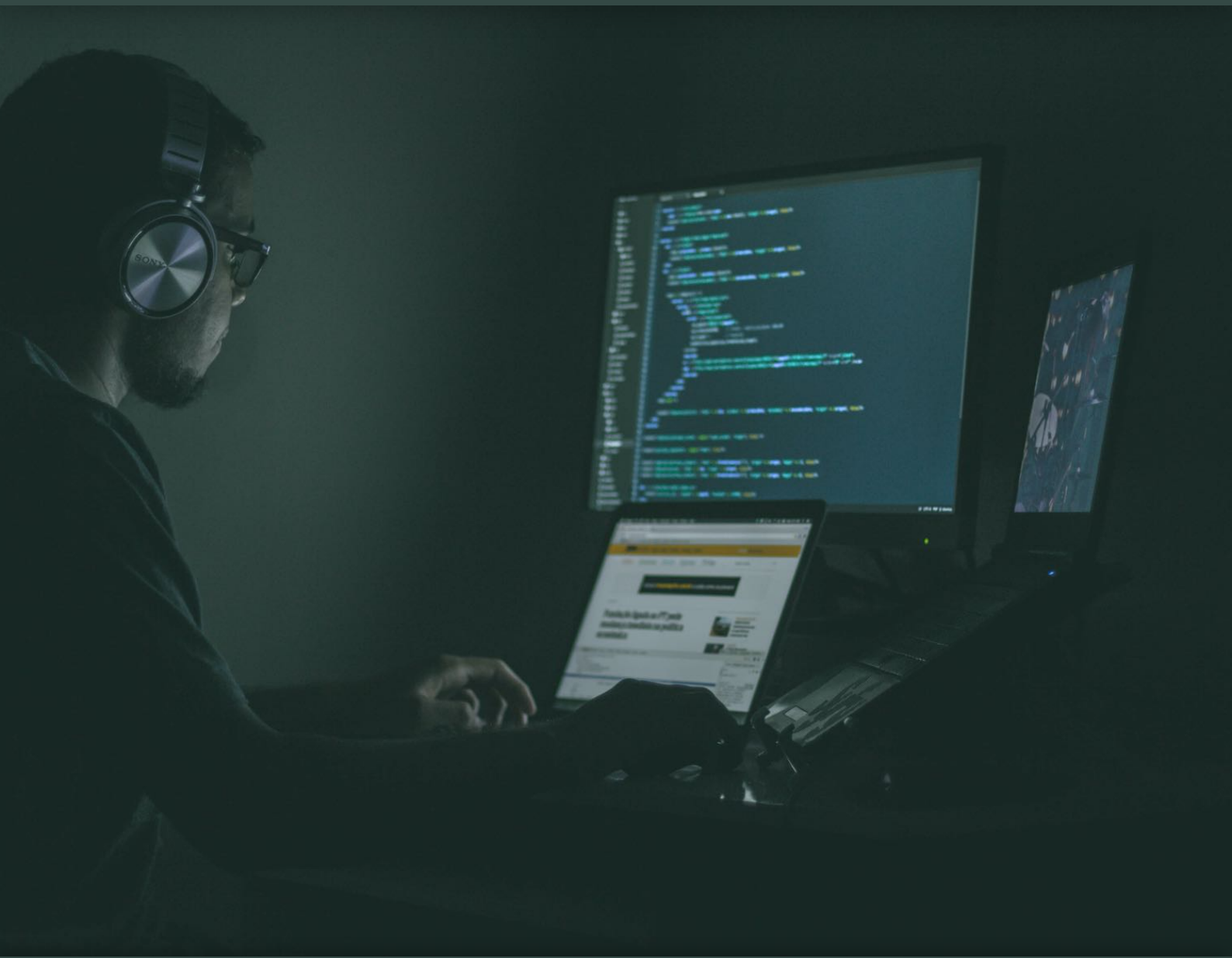
Αρχές Γλωσσών Προγραμματισμού και Μεταφραστών



Απαντήσεις Project

Αθανασία Ζεκυριά (1059660) — Λουδάρος Ιωάννης (1067400)
Τσίκελης Ιωάννης (1067407) — Χριστίνα Κρατημένου (1067495)

Γενικές Πληροφορίες

Στις επόμενες σελίδες αναλύονται οι απαντήσεις τις ομάδας μας στο Project του μαθήματος “Αρχές Γλωσσών Προγραμματισμού και Μεταφραστών”. Σε αυτή την σελίδα έχετε πρόσβαση σε γενικές πληροφορίες γύρω από το Project αλλά και γύρω από τα μέλη της ομάδας μας.

Για να έχετε πρόσβαση στην τελευταία έκδοση των απαντήσεων μπορείτε να σκανάρετε το παραπάνω QR Code ή να χρησιμοποιήσετε το παρακάτω κουμπί.

Πατήστε Εδώ

Για την υλοποίηση του Project εργαστήκαμε σε μια ομάδα 4 ατόμων. Κάναμε συναντήσεις σε εβδομαδιαία βάση για να αξιολογήσουμε την πρόοδο μας αλλά και να καθορίσουμε τους επόμενους μας στόχους. Όλοι είχαμε επίγνωση των εργασιών που πραγματοποιούν οι συνεργάτες μας και δίναμε feedback ο ένας στον άλλον με αποτέλεσμα να προχωράμε συνεκτικά. Η ομάδα αποτελείται από τα εξής άτομα:

Ιωάννης Λουδάρος (1067400)
Χριστίνα Κρατημένου (1067495)
Αθανασία Ζεκυριά (1059660)
Ιωάννης Τσικέλης (1067407)

Παρακάτω υπάρχουν αναλυτικότερες πληροφορίες για τα μέλη της ομάδας μας.

Ιωάννης Τσικέλης
1067407

st1067407@ceid.upatras.gr
Φοιτητής 3ου έτους

Αθανασία Ζεκυριά
1059660

up1059660@upnet.gr
Φοιτήτρια 4ου έτους

Ιωάννης Λουδάρος
1067400

iloudaros@upnet.gr
Φοιτητής 3ου έτους

Χριστίνα Κρατημένου
1067495

up1067495@upnet.gr
Φοιτήτρια 3ου έτους

Περιεχόμενα

Συντακτικός ορισμός της Γλώσσας σε BNF

Θεμελιώδεις έννοιες

Στις θεμελιώδεις έννοιες ορίζουμε τα κύτταρα που απαρτίζουν ένα πρόγραμμα. Σε αυτά συμπεριλαμβάνονται τα integers, τα ονόματα των μεταβλητών και άλλα.

Τύποι μεταβλητών

```
<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
<integer> ::= [ - ] ( <digit> | <digit> <integer> )
<p-integer> ::= <digit> | <digit> <integer>
<char> ::= a | ... | z | A | ... | Z
<punctuation> ::= .
                | ?
                | !
                | ,
                | ;
                | :
                | -
                | "["
                | "]"
                | "{"
                | "}"
                | "
                | "
                | "("
                | ")"
                | '
<string> ::= <letter> <string>
           | <digit> <string>
           | <punctuation> <string>
           | <operator> <string>
           | <letter>
           | <digit>
<var-name> ::= ( ( _ | <letter> ) [<var-name>] )
             | <letter> <var-name>
             | <digit> <var-name>
             | _ <var-name>
             | <letter>
             | <digit>
<variable> ::= <type> <var-name> [ "[" <p-integer> "]" ]
<variables> ::= <variable> | <variable>, <variables>
```

Οι μεταβλητές ακολουθούν την ονοματολογία της C.

Operators

```
<operator> ::= <a-operator> | <l-operator>
<a-operator> ::= +
               | -
               | ^
               | *
               | /
```

Ξεκινάμε να ορίζουμε τους τελεστές που αναγνωρίζει η HotSauce.

```
<l-operator> ::= AND
                | OR
                | ==
                | !=
                | <
                | >
```

Keywords and Special Characters

```
<ws> ::= \t | \n | " "
<type> ::= INT | CHAR
```

Δομικά block ενός προγράμματος

Δηλώσεις

```
<program title> ::= PROGRAM <string> \n
<func-declaration> ::= FUNCTION <string> "(" <variables> ")"
<var-declaration> ::= VARS <variables> ;
```

Εντολές

```
<a-expression> ::= <integer>
                | <var-name>
                | <a-expression> <a-operator> <a-expression>
                | "(" <a-expression> ")"
                | - <a-expression>
<l-expression> ::= <integer>
                | <var-name>
                | <a-expression>
                | <l-expression> <l-operator> <l-expression>
                | "(" <l-expression> ")"
<assignment> ::= ( <var-name> ) = ( <a-expression> | <char> ) ;
<print> ::= PRINT "(" <string> [ , "[" <var-name> "]" ] ")" ;
```

Δομές Επανάληψης

```
<while> ::= WHILE "(" <l-expression> ")" \n
            <statements> \n
            ENDWHILE
<for> ::= FOR counter:= <integer> TO <integer> STEP <integer> \n
            <statements> \n
           ENDFOR
```

Δομές Απόφασης

```
<if> ::= IF "(" <l-expression> ")" THEN \n
          <statements> \n
          { ELSEIF \n
           <statements> \n }
          [ ELSE <statements> \n ]
           ENDIF
<switch> ::= SWITCH "(" <l-expression> ")" \n
CASE "(" <l-expression> ")" : \n
 <statements> \n
```

Τα Δομικά block ενός προγράμματος είναι δομές και εντολές οι οποίες πρέπει να ακολουθούν κανόνες. Τέτοια είναι οι δηλώσεις, οι δομές επανάληψης και απόφασης κλπ.

{ CASE “(“ <l-expression> “)“ : \n
<statements> \n }
[DEFAULT: \n
<statements> \n]
ENDSWITCH

Σχόλια
<commend> ::= % <string>

Πιο αφηρημένες έννοιες

<statements> ::= <assignment> [<statements>]
| <print> [<statements>]
| <while> [<statements>]
| <for> [<statements>]
| <if> [<statements>]
| <switch> [<statements>]
| <commend> [<statements>]
| BREAK;
| <string> “(“ <variables> “)“ \n
“{“ <statements> “}“
| “{“ <statements> “}“
<function> ::= <func-declaration> \n
<var-declaration> \n
<statements> \n
RETURN (<a-expression> | <string>) \n
END_FUNCTION
<main> ::= STARTMAIN \n
<var-declaration> \n
<statements>
ENDMAIN
<program> ::= <program title> { <function> } <main>

Υποσημείωση : Σε κάποια σημεία είναι απαραίτητο, μέσα σε παραγωγές να χρησιμοποιήσουμε σύμβολα που κανονικά είναι δεσμευμένα για την σημειογραφία Backus-Naur. Σε αυτή την περίπτωση αντί για απλή αναγραφή του στοιχείου, περικλείεται σε κόκκινα εισαγωγικά (πχ “(“)

Οι πιο αφηρημένες έννοιες περιγράφουν την δομή ολόκληρου του προγράμματος.

Αυτή την γραμμή την χρησιμοποιούμε ώστε ο χρήστης να μπορεί να χρησιμοποιεί τυχόν συναρτήσεις που ορίζει.

Lexical Analysis

./Code/lexical_analysis_flex/hotsauce.l

Γενικά για το Lexer

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

%option noyywrap
%option yylineno

identifier [a-zA-Z_][a-zA-Z_$0-9]*

digit [0-9]
non_zero_digit [1-9]
number {non_zero_digit}{digit}*|"0"

string_item ([^\n\'\"\\])|([\\].)
string   (['']{string_item}*['])|(["]
{string_item}*["])

whitespace    [ \t\v\n\f]
```

Στις πρώτες γραμμές του “hotsauce.l” κάνουμε τα κατάλληλα include για το project, συμπεριλαμβανομένου του “hotsauce.tab.h” από το οποίο θα παρθούν τα tokens.

Σε αυτό το σημείο ορίζουμε το identifier. Το χρησιμοποιούμε αργότερα για να αντιλαμβανόμαστε ονόματα μεταβλητών και συναρτήσεων.

Ο λόγος ύπαρξης αυτού του κανόνα είναι η μη αποδοχή αριθμών όπως “010” μιας

Ορίζουμε επίσης στην αρχή τα βασικά patterns που θα χρησιμοποιήσουμε αργότερα στους κανόνες του lexer.

Στις επόμενες γραμμές ορίζουμε τα απαραίτητα tokens αλλά και τους operators που καταλαβαίνει η HotSauce.

```
{string}        {
                  if(strlen(yytext) > 3) return KEY_STRING;
                  else return KEY_CHARACTER;
                }
```

Ενδιαφέρον έχουν οι γραμμές 81-84 (όπως φαίνονται παραπάνω) όπου ορίζεται ο τρόπος που το lexer διαχωρίζει έναν χαρακτήρα από ένα string.

Comments

Παρακάτω βλέπουμε τον τρόπο που υλοποιούμε τα comments. Αναγνωρίζονται κατευθείαν από το lexer και αγνοούνται.

```
[%].*                   { /*Ignore comments*/}
[/][*][^*]*[*]+([^*/][^*]*[*]+)*[/] { /*Ignore multiline comments*/}
```

Τα multiline comments, θα μπορούσαν απλά να αναγνωρίζονται από την παραπάνω κανονική έκφραση, παρόλα αυτά, η εκφώνηση ζητάει ρητά να μην χρησιμοποιηθεί μια τέτοια τεχνική. Για αυτό δημιουργούμε την συνάρτηση skip_multiple_line_comment( ) η οποία καλείται κάθε φορά που το scanner συναντάει ένα “/*”. Δουλειά της συνάρτησης αυτής είναι να καταναλώνει χαρακτήρες της εισόδου μέχρι να συναντήσει το “*/“. Αν στο μεταξύ συναντήσει το EOF πριν από τότε, τυπώνει μήνυμα λάθους.

```
"/*"            { puts("long comment begins ");
                  skip_multiple_line_comment();
                  puts("long comment ends");
                 }
```

```
static void skip_multiple_line_comment(void)
{
  int c;

  for(;;)
  {
    switch(input())
    {
      /* We expect ending the comment first before EOF */
      case EOF:
        fprintf(stderr, "Error unclosed comment, expect */\n");
        exit(-1);
        goto done;
      break;
      /* Is it the end of comment? */
      case '*':
        if((c = input()) == '/') goto done;
        unput(c);
        break;
      default:
        /* skip this character */
        break;
    }
  }

done:
  /* exit entry */ ;
}
```

Syntax Analysis

./Code/syntax_analysis_bison/hotsauce.y

Γενικά για το Parser

```
%{

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

//Declarations of flex derived tools

extern int yylex();
extern int yyparse();
extern FILE *yyin;
extern FILE *yyout;
extern int yylineno;
extern int  yywrap;

int yylex();
void yyerror(const char* s);

%}
```

Στην αρχή του “hotsauce.y” ηλώνουμε τα εργαλεία που θα χρησιμοποιήσουμε μέσα στο parser μας.

```
/*** Here, we declare the tokens ***/

%token KEY_PROGRAM
%token KEY_FUNCTION KEY_RETURN KEY_ENDFUNCTION
%token KEY_VARS KEY_CHAR KEY_INT

...

%left KEY_MUL KEY_DIV
%left KEY_PLUS KEY_MIN

%token KEY_TYPEDEF
%token KEY_STRUCT
%token KEY_ENDSTRUCT
```

Στις γραμμές 22 με 50 δηλώνουμε τα keys αλλά και τα operators -καθώς και την προτεραιότητα τους-που θα χρησιμοποιήσουμε.

```
/*** Here, you can see the rules ***/

//Declarations
program:
  KEY_PROGRAM KEY_IDENTIFIER struct_decls
functions main {printf("Parsed Successfully!
🔥");}
  ;
```

Από την γραμμή 56 ξεκινάνε οι κανόνες που ορίζουν το συντακτικό του parser μας.

Δίπλα παραθέτουμε τον κανόνα που παράγει την κορυφή του συντακτικού μας δέντρου. Μόλις τελειώσει η αναγνώριση του, σημαίνει ότι έχει ολοκληρωθεί το πρόγραμμα μας, για αυτό και εκτυπώνει μήνυμα επιτυχίας.

```
void yyerror(const char* s)
  {
     fprintf(stderr, "Line: %d --> Parser
error\n", yylineno);
     exit(1);
  }
```

Εδώ βλέπουμε την υλοποίηση της yyerror( ). Χρησιμοποιείται το yylineno ώστε να μαθαίνουμε την γραμμή στην οποία προέκυψε σφάλμα.
Η yyerror( ) ορίζεται στην γραμμή 313.

```
int main ( int argc, char **argv  )
  {

  yyin = fopen( argv[1], "r" );
  yyparse ();

  return 0;
  }
```

Η main το μόνο που κάνει είναι να ορίζει ότι το scanner πρέπει να ξεκινήσει από το αρχείο το οποίο δίνουμε ως option στην κλήση του εκτελέσημου και να καλεί την yyparse( ).

./Code/declaration_controller

Τεχνική Ελέγχου Ύπαρξης Μεταβλητών, Συναρτήσεων Και Structs

Οι κανόνες οι οποίοι αναλαμβάνουν τις δηλώσεις των Μεταβλητών, των Συναρτήσεων ή των Structs, κρατάνε κάθε φορά το όνομα του αντίστοιχου identifier μέσα σε μια από τις 3 λίστες που αντιστοιχούν στο κάθε είδος δήλωσης (χρήση declare( )). Αργότερα, οι κανόνες οι οποίοι χειρίζονται identifiers που είναι ονόματα Μεταβλητών, Συναρτήσεων ή Structs, ελέγχουν αν το identifier που αναγνώρισαν υπάρχει στην αντίστοιχη λίστα (χρήση check_arr( )). Αν υπάρχει, σημαίνει ότι είχε δηλωθεί και συνεπώς η χρήση του είναι αποδεκτή, διαφορετικά το hotsauce τερματίζει.

```
identifier_list:
  KEY_IDENTIFIER {declared_vars.size++;
declare(&declared_vars,$1);}
  | identifier_list KEY_COMMA KEY_IDENTIFIER
  ;
```

Εδώ βλέπουμε έναν κανόνα που χρησιμοποιείται κατά την δήλωση μεταβλητών και κάνει χρήση του declare( ).

```
statement:
  assignment
  | while
  | for
  | if
  | switch
  | print
  | KEY_IDENTIFIER KEY_PARL
identifier_list KEY_PARR KEY_SEMICOLON
{ check_arr(&declared_funcs,$1); }
  | KEY_IDENTIFIER KEY_PARL KEY_PARR
KEY_SEMICOLON { check_arr(&declared_funcs,
$1); }
    ;
```

Εδώ αντίστοιχα βλέπουμε τι συμβαίνει όταν ο parser συναντάει την κλήση μιας συνάρτησης. Χρησιμοποιείται η check_arr( ) για να βεβαιωθούμε ότι έχει προηγηθεί δήλωση της συνάρτησης.

```
typedef struct{
  char** list;
  int size;
}log;


void declare(log* l, char* identifier);


void check_arr(log* l, char*
identifier);
```

Αξίζει επίσης να σημειωθεί ότι για την υλοποίηση των λιστών και για να διατηρούμε τον κώδικα όσο πιο ευανάγνωστο γίνεται, ορίσαμε τον τύπο log ο οποίος περιλαμβάνει τον δυναμικό πίνακα που αποθηκεύει τα ονόματα των identifiers και το τρέχον μέγεθος του.

Εκτέλεση του Hotsauce

```
./Code/Compiler
```

Compilation και Εκτέλεση

Αρκεί μια εντολή make στο συγκεκριμένο directory για να γίνει το compilation. Μετά την ολοκλήρωση του, εμφανίζεται η επιλογή στον χρήστη να χρησιμοποιήσει το hotsauce για να κάνει “compile” το test code που υπάρχει στο ./Code/test_code/test.hotsauce

```
./Code/exec_helper
```

---

exec_helper

Το make, καλεί το exec_helper, ένα βοηθητικό πρόγραμμα το οποίο απλά φροντίζει την διεπαφή που έχει ο χρήστης μετά την ολοκλήρωση του compilation του hotsauce.