**Αρχές Γλωσσών Προγραμματισμού & Μεταφραστών**

**Τμήμα Μηχανικών Η/Υ & Πληροφορικής**

**Πανεπιστήμιο Πατρών**

**Εαρινό Εξάμηνο 2021**

**Διδάσκοντες: Ι. Γαροφαλάκης, Σ. Σιούτας**

**Εργαστηριακή Άσκηση**

**Τα στοιχεία των μελών της ομάδας:**

|  |  |  |
| --- | --- | --- |
| Ονοματεπώνυμο | AM | Email |
| Αβραμόπουλος Γεώργιος | 1070772 | up1070772@upnet.gr |
| Νικόλαος Κοντογιώργης | 1070922 | up1070922@upnet.gr |
| Ορέστης Μαζαρακιώτης | 1064028 | up1064028@upnet.gr |
| Νεφέλη Κωνσταντοπούλου | 1059628 | up1059628@upnet.gr |

**1. Περιγραφή της γραμματικής της γλώσσας σε BNF:**

**<A\_PARENTHESI> ::= “(”**

**<D\_PARENTHESI> ::= “)”**

**<KOMMA> ::= “,”**

**<Q\_MARK> :: “;”**

**<A\_BRACKET> ::= “[”**

**<D\_BRACKET> ::= “]”**

**<ADD> ::= “+”**

**<SUBTRACT> ::= “-”**

**<POWER\_OF> ::= “^”**

**<MULTIPLY> ::= “\*”**

**<DIVIDE> ::= “/”**

**<EQUALS> ::= “=”**

**<BIGGER\_THAN> ::= “<”**

**<SMALLER\_THAN> ::= “>”**

**<LOG\_EQUALS> ::= “==”**

**<NOT\_EQUAL> ::= “!=”**

**<COLON> ::= “:”**

**<ARIST> ::= “’”**

**<WHILE> ::= “while”**

**<ENDWHILE> ::= “end\_while”**

**<AND> ::= “and”**

**<OR> ::= “or”**

**<FOR> ::= “for”**

**<COUNTER> ::= “counter”**

**<TO> ::= “to”**

**<STEP> ::= “step”**

**<ENDFOR> ::= “end\_for”**

**<IF> ::= “if”**

**<THEN> ::= “then”**

**<ELSE> ::= “else”**

**<ELSEIF> ::= “else\_if”**

**<ENDIF> ::= “end\_if”**

**<SWITCH> ::= “switch”**

**<CASE> ::= “case”**

**<DEFAULT> ::= “default”**

**<END\_SWITCH> ::= “end\_switch”**

**<PRINT> ::= “print”**

**<BREAK> ::= “break”**

**<PROGRAM> ::= “program”**

**<RETURN> ::= “return”**

**<THETIKOS\_AKER> ::= “thetikos\_aker”**

**<FLOAT> ::= “myfloat”**

**<STARTMAIN> ::= “start\_main”**

**<ENDMAIN> ::= “end\_main”**

**<FUNCTION> ::= “function”**

**<VARS> ::= “vars”**

**<END\_FUNCTION> ::= “end\_function”**

**<STRUCT> ::= “struct”**

**<ENDSTRUCT> ::= “endstruct”**

**<TYPEDEF> ::= “typedef”**

**<CHAR> ::= “mychar”**

**<INTEGER> ::= “myinteger”**

**<LETTER> ::= 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 |**

**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**

**<STRING> ::= <LETTER> |**

**<LETTER><STRING> |**

**<DIGIT><STRING>**

**<main> ::= <program><list\_structs><list\_function><function><startmain>**

**| <program><list\_structs><startmain>**

**| <program><startmain>**

**| <program>**

**<program> ::= <PROGRAM><STRING>**

**<list\_structs> ::= <list\_structs><structs>**

**| <structs>**

**<structs> ::= <STRUCT><STRING><vars><ENDSTRUCT>**

**| <STRUCT><STRING><vars><STRING><ENDSTRUCT>**

**<vars> ::= <VARS><vars\_list>**

**<vars\_list> ::= <CHAR><string\_list><Q\_MARK><vars\_list>**

**| <INTEGER><string\_list><Q\_MARK><vars\_list>**

**| <CHAR><string\_list><Q\_MARK>**

**| <INTEGER><string\_list><Q\_MARK>**

**<string\_list> ::=<STRING><A\_BRACKET><THETIKOS\_AKER><D\_BRACKET><KOMMA>**

**<string\_list>**

**| <STRING><KOMMA><string\_list>**

**| <STRING><A\_BRACKET><THETIKOS\_AKER><D\_BRACKET>**

**| <STRING>**

**<list\_function> ::= <list\_function><function>**

**| <function>**

**<function> ::= <FUNCTION><STRING><A\_PARENTHESI><string\_list><D\_PARENTHESI><vars>**

**<entoles><return><END\_FUNCTION>**

**| <FUNCTION><STRING><A\_PARENTHESI><D\_PARENTHESI><vars><entoles><return>**

**<END\_FUNCTION>**

**| <FUNCTION><STRING><A\_PARENTHESI><string\_list><D\_PARENTHESI><entoles>**

**<return><END\_FUNCTION>**

**<return> ::= <RETURN><THETIKOS\_AKER><Q\_MARK>**

**| <RETURN><STRING><Q\_MARK>**

**<list\_entoles> ::= <list\_entoles><entoles>**

**| <entoles>**

**<entoles> ::= <anathesi>**

**| <while>**

**| <for>**

**| <if>**

**| <switch>**

**| <print>**

**| <break>**

**<print> ::= <PRINT><A\_PARENTHESI><ARIST><STRING><ARIST><D\_PARENTHESI><Q\_MARK>**

**| <PRINT><A\_PARENTHESI><ARIST><STRING><ARIST><KOMMA><string\_list>**

**<Q\_MARK>**

**<break> ::= <BREAK><Q\_MARK>**

**<anathesi> ::= <STRING><EQUALS><prajeis><Q\_MARK>**

**<prajeis> ::= <prajeis><ADD><prajeis>**

**| <prajeis><SUBTRACT><prajeis>**

**| <prajeis><MULTIPLY><prajeis>**

**| <prajeis><DIVIDE><prajeis>**

**| <prajeis><POWER\_OF><prajeis>**

**| <A\_PARENTHESI><prajeis><D\_PARENTHESI>**

**| <STRING><ADD><prajeis>**

**| <STRING><A\_PARENTHESI><string\_list><D\_PARENTHESI><ADD><prajeis>**

**| <THETIKOS\_AKER><ADD><prajeis>**

**| <STRING><SUBTRACT><prajeis>**

**| <THETIKOS\_AKER><SUBTRACT><prajeis>**

**| <STRING><MULTIPLY><prajeis>**

**| <STRING><A\_PARENTHESI><string\_list><D\_PARENTHESI><POWER\_OF><prajeis>**

**| <THETIKOS\_AKER><POWER\_OF><prajeis>**

**| <STRING><A\_PARENTHESI><string\_list><D\_PARENTHESI>**

**| <STRING><A\_PARENTHESI><D\_PARENTHESI>**

**| <THETIKOS\_AKER>**

**<while> ::= <WHILE><condition><list\_entoles><ENDWHILE>**

**<for> ::= <FOR><STRING><COLON><EQUALS><THETIKOS\_AKER><TO><THETIKOS\_AKER>**

**<STEP><THETIKOS\_AKER><ENDFOR>**

**<list\_elseif> ::= <list\_elseif><elseif>**

**| <elseif>**

**<elseif> ::= <ELSEIF><condition><list\_entoles>**

**<if> ::= <IF><condition><THEN><list\_entoles><ENDIF>**

**| <IF><condition><THEN><list\_entoles><ELSE><list\_entoles><ENDIF>**

**| <IF><condition><THEN><list\_entoles><list\_elseif><ENDIF>**

**| <IF><condition><THEN><list\_entoles><list\_elseif><ELSE><list\_entoles><ENDIF>**

**<list\_case> ::= <list\_case><case>**

**| <case>**

**<case> ::= <CASE><prajeis><COLON><list\_entoles>**

**<switch> ::= <SWITCH><prajeis><list\_case><DEFAULT><COLON><list\_entoles><ENDSWITCH>**

**| <SWITCH><prajeis><list\_case><ENDSWITCH>**

**<and\_or> ::= <AND>**

**| <AND><and\_or>**

**| <OR>**

**| <OR><and\_or>**

**<condition> ::= <A\_PARENTHESI><STRING><BIGGER\_THAN><THETIKOS\_AKER>**

**<D\_PARENTHESI>**

**| <A\_PARENTHESI><STRING><BIGGER\_THAN><THETIKOS\_AKER>**

**<D\_PARENTHESI><and\_or><condition>**

**| <A\_PARENTHESI><STRING><BIGGER\_THAN><STRING><D\_PARENTHESI>**

**| <A\_PARENTHESI><STRING><BIGGER\_THAN><STRING><D\_PARENTHESI>**

**<and\_or><condition>**

**| <A\_PARENTHESI><STRING><SMALLER\_THAN><THETIKOS\_AKER>**

**<D\_PARENTHESI>**

**| <A\_PARENTHESI><STRING><SMALLER\_THAN><THETIKOS\_AKER>**

**<D\_PARENTHESI><and\_or><condition>**

**| <A\_PARENTHESI><STRING><SMALLER\_THAN><STRING><D\_PARENTHESI>**

**| <A\_PARENTHESI><STRING><SMALLER\_THAN><STRING><D\_PARENTHESI>**

**<and\_or><condition>**

**| <A\_PARENTHESI><STRING><NOT\_EQUAL><THETIKOS\_AKER><D\_PARENTHESI>**

**| <A\_PARENTHESI><STRING><NOT\_EQUAL><THETIKOS\_AKER><D\_PARENTHESI>**

**<and\_or><condition>**

**<A\_PARENTHESI><condition><D\_PARENTHESI>**

**<startmain> ::= <STARTMAIN><vars><list\_entoles><ENDMAIN>**

**| <STARTMAIN><list\_entoles><ENDMAIN>**

**2.Αρχείο FLEX**

%option noyywrap

%{

#include <stdio.h> //tomeas orismwn

#include <stdlib.h>

#include "project.tab.h" //diasindesi flex kai bison arxeiou

//EXEI SIMASIA ME POIA SEIRA TA GRAFOYME TA PARAKATO YPOSINOLO PIO PSILA APO YPERSINOLO

//idia seira me kanones isos na doyleyei sosta

extern int yylex();

%}

%option noyywrap

comment ^[%]+.\*

program (PROGRAM)

function (FUNCTION)

vars (VARS)

mychar (CHAR)

myinteger (INTEGER)

end\_function (END\_FUNCTION)

start\_main (STARTMAIN)

end\_main (ENDMAIN)

return (RETURN)

while (WHILE)

end\_while (ENDWHILE)

and (AND)

or (OR)

for (FOR)

counter (counter:=)

to (TO)

step (STEP)

end\_for (ENDFOR)

if (IF)

then (THEN)

else (ELSE)

else\_if (ELSEIF)

end\_if (ENDIF)

switch (SWITCH)

case (CASE)

default (DEFAULT)

end\_switch (ENDSWITCH)

print (PRINT)

break (BREAK)

struct (STRUCT)

endstruct (ENDSTRUCT)

typedef (TYPEDEF)

thetikos\_aker [1-9]+

myfloat [0-9]+\.[0-9]+

mystring [A-Za-z]+[A-Za-z0-9]\*

%%

{comment} {};

"(" return A\_PARENTHESI;

")" return D\_PARENTHESI;

"," return KOMMA;

";" return Q\_MARK;

"[" return A\_BRACKET;

"]" return D\_BRACKET;

"+" return ADD;

"-" return SUBTRACT;

"^" return POWER\_OF;

"\*" return MULTIPLY;

"/" return DIVIDE;

"=" return EQUALS;

">" return BIGGER\_THAN;

"<" return SMALLER\_THAN;

"==" return LOG\_EQUALS;

"!=" return NOT\_EQUAL;

":" return COLON;

"'" return ARIST;

{while} return WHILE;

{end\_while} return ENDWHILE;

{and} return AND;

{or} return OR;

{for} return FOR;

{counter} return COUNTER;

{to} return TO;

{step} return STEP;

{end\_for} return ENDFOR;

{if} return IF;

{then} return THEN;

{else} return ELSE;

{else\_if} return ELSEIF;

{end\_if} return ENDIF;

{switch} return SWITCH;

{case} return CASE;

{default} return DEFAULT;

{end\_switch} return ENDSWITCH;

{print} return PRINT;

{break} return BREAK;

{program} return PROGRAM;

{return} return RETURN;

{thetikos\_aker} return THETIKOS\_AKER;

{myfloat} return FLOAT;

{start\_main} return STARTMAIN;

{end\_main} return ENDMAIN;

{function} return FUNCTION;

{vars} return VARS;

{end\_function} return END\_FUNCTION;

{struct} return STRUCT;

{endstruct} return ENDSTRUCT;

{typedef} return TYPEDEF;

{mychar} return CHAR;

{myinteger} return INTEGER;

{mystring} {yylval.sval = strdup(yytext); return STRING;}

%%

**3. Αρχείο Bison**

%{

#include <stdio.h> **//απαραίτητες βιβλιοθήκες**

#include <stdlib.h>

#include <string.h>

extern int yylex(); //

extern int yyparse();

extern FILE \*yyin; **//μεταβλητές για είσοδο και έξοδο αρχείου**

void yyerror(const char \*s);

typedef struct metabliti{

char onoma[32];

} metabliti;

metabliti\* structs\_list;

int structs\_count;

void newstruct(char\* str) {

structs\_count += 1;

structs\_list = calloc(structs\_count, sizeof(metabliti));

strcpy(structs\_list[structs\_count - 1].onoma, str);

printf("\n\n\n%d\n\n\n\n", structs\_count);

printf("\n%s\n\n", (structs\_list[structs\_count - 1].onoma)) }

void foundstruct(char\* str){

int flag = 0;

for (int i = 0; i < structs\_count; i++){

if (!strcmp(str, structs\_list[i].onoma)){

flag = 1;

break; } }

if (flag == 1){

printf("\n\nto onoma sinartisis iparxei idi\n\n");

} else{

printf("\n\nden einai dilomeni h: %s\n\n", str);

exit(1); } }

%}

%union { **//union που ορίσαμε τα types των διαφόρων tokens**

char\* sval;}

%token FLOAT;

%token PROGRAM;

%token <sval> STRING;

%token FUNCTION;

%token VARS;

%token CHAR;

%token INTEGER;

%token A\_PARENTHESI;

%token D\_PARENTHESI;

%token KOMMA;

%token Q\_MARK;

%token THETIKOS\_AKER;

%token A\_BRACKET;

%token D\_BRACKET;

%token END\_FUNCTION;

%token RETURN;

%token STARTMAIN;

%token ENDMAIN;

%token ADD;

%token SUBTRACT;

%token POWER\_OF;

%token MULTIPLY;

%token DIVIDE;

%token EQUALS;

%token WHILE;

%token ENDWHILE;

%token BIGGER\_THAN;

%token SMALLER\_THAN;

%token LOG\_EQUALS;

%token NOT\_EQUAL;

%token AND;

%token OR;

%token FOR;

%token COUNTER;

%token TO;

%token STEP;

%token ENDFOR;

%token IF;

%token THEN;

%token ELSE;

%token ELSEIF;

%token ENDIF;

%token SWITCH;

%token CASE;

%token COLON;

%token DEFAULT;

%token ENDSWITCH;

%token PRINT;

%token BREAK;

%token ARIST;

%token STRUCT;

%token ENDSTRUCT;

%token TYPEDEF;

%%

main: program structs function entoles function\_end code{

printf("\nfoyl komple\n\n");}

| program {

printf("\nfoyl komple allo\n\n"); } ;

program: PROGRAM STRING {

printf("\nprogramma komple\n\n"); };

structs: STRUCT STRING VARS CHAR STRING Q\_MARK ENDSTRUCT{

printf("\n\n======= VRHKA TO: %s\n\n", $2);

newstruct($2);}

| STRUCT STRING VARS CHAR STRING A\_BRACKET THETIKOS\_AKER D\_BRACKET Q\_MARK ENDSTRUCT{

newstruct($2);}

| STRUCT STRING VARS CHAR THETIKOS\_AKER Q\_MARK ENDSTRUCT{

newstruct($2) }

| TYPEDEF STRUCT STRING VARS CHAR STRING Q\_MARK STRING ENDSTRUCT{

printf("\n\n======= VRHKA TO: %s\n\n", $3);

newstruct($3); }

| TYPEDEF STRUCT STRING VARS CHAR STRING A\_BRACKET THETIKOS\_AKER D\_BRACKET Q\_MARK STRING ENDSTRUCT{

newstruct($3); }

| TYPEDEF STRUCT STRING VARS CHAR THETIKOS\_AKER Q\_MARK STRING ENDSTRUCT{

newstruct($3); };

function: FUNCTION STRING A\_PARENTHESI list D\_PARENTHESI function\_body{

printf("\nfunction komple\n\n");}

| FUNCTION STRING function\_body {

printf("\nfunction xoris parameters komple\n\n"); };

code: STARTMAIN function\_body ENDMAIN{

printf("\ncode komple\n\n");}

| STARTMAIN ENDMAIN {

printf("\ncode komple xoris metablites\n\n"); }

| STARTMAIN function\_body entoles ENDMAIN {

printf("\ncode komple me metablites kai entoles\n\n"); }

| STARTMAIN entoles ENDMAIN {

printf("\ncode komple xoris metablites me entoles\n\n"); }

| STARTMAIN function\_body entoles a\_entoles ENDMAIN {

printf("\ncode komple me metablites kai entoles kai entoles programmatos\n\n"); };

a\_entoles: p\_entoles

| p\_entoles a\_entoles

| e\_entoles

| e\_entoles a\_entoles;

p\_entoles: entoles\_while

| entoles\_while p\_entoles

| entoles\_for

| entoles\_for p\_entoles;

e\_entoles: entoles\_if

| entoles\_if e\_entoles

| entoles\_switch

| entoles\_switch e\_entoles;

case\_s: CASE prajeis COLON entoles

| CASE prajeis COLON entoles case\_s;

entoles\_switch: SWITCH prajeis case\_s DEFAULT COLON entoles ENDSWITCH{

printf("\nswitch\n\n");}

| SWITCH prajeis case\_s ENDSWITCH{

printf("\nswitch\n\n"); };

if\_comp: ELSEIF condition

| ELSE;

entoles\_if: if\_comp entoles entoles\_if{

printf("\nelseif or else\n\n");}

| IF condition THEN entoles{

printf("\nif\n\n"); }

| IF condition THEN entoles entoles\_if{

printf("\nif\n\n");}

| ENDIF;

entoles\_for: FOR COUNTER THETIKOS\_AKER TO THETIKOS\_AKER STEP THETIKOS\_AKER entoles ENDFOR{

printf("\nfor loop\n\n");}

| FOR COUNTER THETIKOS\_AKER TO THETIKOS\_AKER STEP THETIKOS\_AKER entoles ENDFOR entoles\_for{

printf("\nfor loop\n\n");};

and\_or: AND

| AND and\_or

| OR

| OR and\_or;

condition: A\_PARENTHESI STRING BIGGER\_THAN THETIKOS\_AKER D\_PARENTHESI

| A\_PARENTHESI STRING BIGGER\_THAN THETIKOS\_AKER D\_PARENTHESI and\_or condition

| A\_PARENTHESI STRING BIGGER\_THAN STRING D\_PARENTHESI

| A\_PARENTHESI STRING BIGGER\_THAN STRING D\_PARENTHESI and\_or condition

| A\_PARENTHESI STRING SMALLER\_THAN THETIKOS\_AKER D\_PARENTHESI

| A\_PARENTHESI STRING SMALLER\_THAN THETIKOS\_AKER D\_PARENTHESI and\_or condition

| A\_PARENTHESI STRING SMALLER\_THAN STRING D\_PARENTHESI

| A\_PARENTHESI STRING SMALLER\_THAN STRING D\_PARENTHESI and\_or condition

| A\_PARENTHESI STRING LOG\_EQUALS THETIKOS\_AKER D\_PARENTHESI

| A\_PARENTHESI STRING LOG\_EQUALS THETIKOS\_AKER D\_PARENTHESI and\_or condition

| A\_PARENTHESI STRING LOG\_EQUALS STRING D\_PARENTHESI

| A\_PARENTHESI STRING LOG\_EQUALS STRING D\_PARENTHESI and\_or condition

| A\_PARENTHESI STRING NOT\_EQUAL THETIKOS\_AKER D\_PARENTHESI

| A\_PARENTHESI STRING NOT\_EQUAL THETIKOS\_AKER D\_PARENTHESI and\_or condition

| A\_PARENTHESI STRING NOT\_EQUAL STRING D\_PARENTHESI

| A\_PARENTHESI STRING NOT\_EQUAL STRING D\_PARENTHESI and\_or condition

| A\_PARENTHESI condition D\_PARENTHESI;

entoles\_while: WHILE condition entoles ENDWHILE {

printf("\nwhile loop\n\n");}

| WHILE condition entoles ENDWHILE p\_entoles {

printf("\nwhile loop\n\n"); } ;

prajeis: prajeis ADD prajeis

| prajeis SUBTRACT prajeis

| prajeis MULTIPLY prajeis

| prajeis DIVIDE prajeis

| prajeis POWER\_OF prajeis

| A\_PARENTHESI prajeis D\_PARENTHESI

| STRING ADD prajeis

| THETIKOS\_AKER ADD prajeis

| STRING SUBTRACT prajeis

| THETIKOS\_AKER SUBTRACT prajeis

| STRING MULTIPLY prajeis

| THETIKOS\_AKER MULTIPLY prajeis

| STRING DIVIDE prajeis

| THETIKOS\_AKER DIVIDE prajeis

| STRING POWER\_OF prajeis

| THETIKOS\_AKER POWER\_OF prajeis

| STRING

| THETIKOS\_AKER;

entoli\_print: PRINT A\_PARENTHESI ARIST STRING ARIST D\_PARENTHESI Q\_MARK

| PRINT A\_PARENTHESI ARIST STRING ARIST KOMMA A\_BRACKET STRING D\_BRACKET D\_PARENTHESI Q\_MARK ;

entoles: STRING EQUALS prajeis Q\_MARK entoles {

foundstruct($1);}

| STRING EQUALS STRING A\_PARENTHESI list D\_PARENTHESI Q\_MARK entoles{

foundstruct($1); }

| STRING EQUALS prajeis Q\_MARK {

foundstruct($1); }

| STRING EQUALS STRING A\_PARENTHESI list D\_PARENTHESI Q\_MARK {

foundstruct($1); }

| BREAK COLON

| entoli\_print;

list: STRING A\_BRACKET THETIKOS\_AKER D\_BRACKET KOMMA list{

newstruct($1);}

| STRING KOMMA list{

newstruct($1); }

| STRING {

newstruct($1); };

parameters: CHAR list Q\_MARK parameters

| INTEGER list Q\_MARK parameters{}

| CHAR list Q\_MARK

| INTEGER list Q\_MARK;

function\_body: VARS parameters {

printf("\nparametroi komple\n\n");};

return: RETURN STRING

| RETURN THETIKOS\_AKER;

function\_end: return END\_FUNCTION {

printf("\nend of function\n\n");}; %%

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

#ifdef YYDEBUG

yydebug = 1;

// yydebug = 0;

#endif

FILE \*myfile = fopen(argv[1], "r");

if (!myfile) {

printf("den yparxei arxeio");

return -1; }

structs\_count = 0;

structs\_list = calloc(structs\_list, 1 \* sizeof(metabliti));

yyin = myfile;

yyparse();}

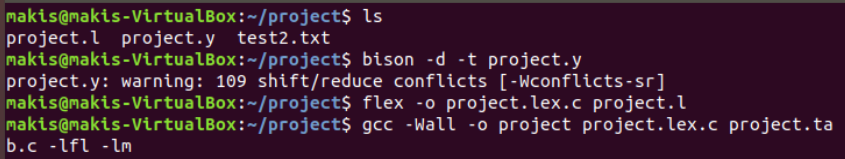
void yyerror(const char \*s) {

return;}

**4. Παραδείγματα Εκτέλεσης Εφαρμογών**

**Τα αρχεία Flex και Bison εκτελέστηκαν σε VirtualBox με λειτουργικο ubuntu.**

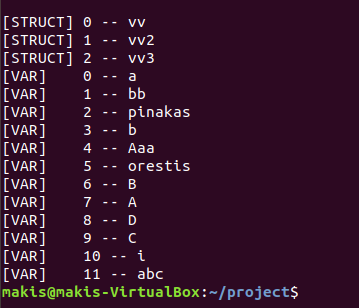
**Στο παρακατω screenshot βλέπουμε τις εντολές που έχουμε εκτελέσει**

****

**Όταν το τρέχουμε χωρίς κάποιο αρχείο**

****

**Με την εντολή ./project test2.txt παίρνουμε την ακόλουθη έξοδο:**



**Στο παρακάτω screenshot εμφανίζονται τα token όταν τρέχουμε το αρχείο test2.txt**

