ΑΡΧΕΣ ΓΛΩΣΣΩΝ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ ΚΑΙ ΜΕΤΑΦΡΑΣΤΩΝ

ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ 2021

ΟΜΑΔΑ

ΜΕΤΣΙ ΦΕΖΟ
ΣΟΥΛΙ ΣΠΥΡΟΣ
ΤΑΜΠΟΥΡΑΤΖΗΣ ΠΑΝΟΣ
ΤΣΑΠΑΡΟΒ ΓΙΑΝΝΗΣ

University of Patras
DEPARTMENT OF COMPUTER AND INFORMATION ENGINEERING
PRINCIPLES OF PROGRAMMING LANGUAGES AND TRANSLATORS
LABORATORY EXERCISE 2021

.

TEAM MEMBERS

METSI FEZO AM: 1070727

E-mail: up1070727@upnet.gr

Year: 3rd

SOULI SPYROS AM: 1070263

E-mail: up1070263@upnet.gr

Year: 4th

TAMPOURATZIS PANOS AM: 1041613

E-mail: ceid5902@upnet.gr

Year: 8th

TSAPAROV YIANNIS AM: 1041620

E-mail: ceid5909@upnet.gr

Year: 8th

Purpose of laboratory exercise

The purpose of the lab exercise was to familiarize ourselves with the description of a language in BNF format, with the basic concepts of compilers and finally to implement two of the parts of a compiler, a syntax and a parser using the Bison and Flex tools. The language parsed into the utterance is a pseudo-language that follows the logic of C.

Question 1

BNF - syntactic definition of pseudolanguage

```
digit  [0-9]
num    {digit}+
letter [a-zA-Z]
char   {letter}|{digit}|_
word   {letter}{char}*

QT     \".*?\"

program: PROGRAM CH NL structs dec_functions main NL

dec_functions: %empty
             |FUNCTION CH args NL body RETURN CH SC NL END_FUNCTION NL
             |FUNCTION CH args NL body RETURN INT SC NL END_FUNCTION NL

struct:       %empty
             |STRUCT CH NL dec_var ENDSTRUCT NL
             |TYPEDEF STRUCT CH NL dec_var CH ENDSTRUCT NL

structs:      struct
             |structs struct

body:        dec_var
             |instructions
             |body instructions

args:        LP vars RP

vars:        INTEGER CH
             | CHAR CH
             | vars CM INTEGER CH
             |vars CM CHAR CH

dec_var:     VARS NL var_dec

var_dec:     %empty
             |INTEGER CH SC NL
             |INTEGER CH SC
             |INTEGER CH ASSIGN_OP INT SC NL
             |INTEGER CH ASSIGN_OP INT SC
             |INTEGER CH LB INT RB SC NL
             |INTEGER CH LB INT RB SC
             |CHAR CH SC NL
             |CHAR CH SC
```

```
            |CHAR CH LB INT RB SC NL
            |CHAR CH LB INT RB SC
            |CHAR CH ASSIGN_OP CH SC NL
            |CHAR CH ASSIGN_OP CH SC
            |CH CH SC NL
            |CH CH SC
            |var_dec CHAR CH SC NL
            |var_dec CHAR CH SC
            |var_dec INTEGER CH SC NL
            |var_dec INTEGER CH SC
            |var_dec INTEGER CH LB INT RB SC NL
            |var_dec INTEGER CH LB INT RB SC
            |var_dec CHAR CH LB INT RB SC NL
            |var_dec CHAR CH LB INT RB SC
            |var_dec INTEGER CH ASSIGN_OP INT SC NL
            |var_dec INTEGER CH ASSIGN_OP INT SC
            |var_dec CHAR CH ASSIGN_OP CH SC NL
            |var_dec CHAR CH ASSIGN_OP CH SC
            |var_dec INTEGER CH ASSIGN_OP math SC NL
            |var_dec INTEGER CH ASSIGN_OP math SC

main:       STARTMAIN NL body ENDMAIN

instructions: instruction instruction
            |instruction instructions

instruction: %empty
            |CH ASSIGN_OP INT SC
            |CH ASSIGN_OP CH SC
            |CH ASSIGN_OP QT SC
            |CH ASSIGN_OP math
            |CH ASSIGN_OP function SC
            |function SC
            |while
            |for
            |print
            |if
            |switch
            |NL

function:   CH LP csv RP
            |CH LP CH RP
            |CH LP INT RP

csv:        CH CM CH
            |INT CM INT
            |CH CM INT
```

```
            |INT CM CH
            |function CM function
            |CH CM function
            |function CM CH
            |INT CM function
            |function CM INT
            |csv CM CH
            |csv CM INT
            |csv CM function

math:       CH"+"CH
            |CH"-"CH
            |CH"*"CH
            |CH"^"CH
            |CH"/"CH
            |INT"+"INT
            |INT"-"INT
            |INT"*"INT
            |INT"^"INT
            |INT"/"INT
            |function"+"function
            |function"-"function
            |function"*"function
            |function"^"function
            |function"/"function
            |function"+"INT
            |function"-"INT
            |function"*"INT
            |function"^"INT
            |function"/"INT
            |function"+"CH
            |function"-"CH
            |function"*"CH
            |function"^"CH
            |function"/"CH
            |CH"+"INT
            |CH"-"INT
            |CH"*"INT
            |CH"^"INT
            |CH"/"INT
            |CH"+"function
            |CH"-"function
            |CH"*"function
            |CH"^"function
            |CH"/"function
            ||INT"+"CH
            ||INT"-"CH
```

```
            |INT'*'CH
            |INT'^'CH
            |INT'/'CH
            |INT'+'function
            |INT'-'function
            |INT'*'function
            |INT'^'function
            |INT'/'function
            |math'+'CH
            |math'-'CH
            |math'*'CH
            |math'^'CH
            |math'/'CH
            |math'+'INT
            |math'-'INT
            |math'*'INT
            |math'^'INT
            |math'/'INT
            |math'+'function
            |math'-'function
            |math'*'function
            |math'^'function
            |math'/'function
            |math SC NL
            |math SC

comp:        CH GT CH
            |CH GT INT
            |CH GT function
            |CH GT LP math RP
            |INT GT INT
            |INT GT CH
            |INT GT function
            |INT GT LP math RP
            |function GT function
            |function GT CH
            |function GT INT
            |function GT LP math RP
            |CH LT CH
            |CH LT INT
            |CH LT function
            |CH LT LP math RP
            |INT LT INT
            |INT LT CH
            |INT LT function
            |INT LT LP math RP
            |function LT function
```

```
            |function LT CH
            |function LT INT
            |function LT LP math RP
            |CH IS_EQ CH
            |CH IS_EQ INT
            |CH IS_EQ function
            |CH IS_EQ LP math RP
            |INT IS_EQ INT
            |INT IS_EQ CH
            |INT IS_EQ function
            |INT IS_EQ LP math RP
            |function IS_EQ function
            |function IS_EQ CH
            |function IS_EQ INT
            |function IS_EQ LP math RP
            |CH ISNOT_EQ CH
            |CH ISNOT_EQ INT
            |CH ISNOT_EQ function
            |CH ISNOT_EQ LP math RP
            |INT ISNOT_EQ INT
            |INT ISNOT_EQ CH
            |INT ISNOT_EQ function
            |INT ISNOT_EQ LP math RP
            |function ISNOT_EQ function
            |function ISNOT_EQ CH
            |function ISNOT_EQ INT
            |function ISNOT_EQ LP math RP
            |comp AND comp
            |comp OR comp

while:      WHILE LP comp RP NL body ENDWHILE NL

for:        FOR CH ASSIGN_OP INT TO INT STEP INT body ENDFOR

print:      PRINT LP QT RP SC
            |PRINT LP QT CM csv RP SC
            |PRINT LP QT CM CH RP SC
            |PRINT LP QT CM INT RP SC
            |PRINT LP QT CM function RP SC

elseif:     ELSEIF LP comp RP THEN
            |elseif body ELSEIF LP comp THEN

if:         IF LP comp RP THEN body ENDIF
            |IF LP comp RP THEN body elseif body ENDIF
            |IF LP comp RP THEN body ELSE body ENDIF
            |IF Lp comp RP THEN body elseif body ELSE body ENDIF
```

```
expr:       INT
            |CH
            |function
            |math

switch:     SWITCH LP expr RP NL case ENDSWITCH
            |SWITCH LP expr RP NL case DEFAULT instructions ENDSWITCH

case:       CASE Lp expr RP CC NL instructions
            |case CASE LP expr RP CC NL instructions
```

lexer.l

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

%option noyywrap
%option yylineno
%x comment

digit   [0-9]
num {digit}+
letter  [a-zA-Z]
char    {letter}|{digit}|_
word    {letter}{char}*

%%

PROGRAM     { yylval = strdup(yytext); return PROGRAM;}
FUNCTION    { yylval = strdup(yytext); return FUNCTION;}
END_FUNCTION    { yylval = strdup(yytext); return END_FUNCTION;}
RETURN      { yylval = strdup(yytext); return RETURN;}
STARTMAIN   { yylval = strdup(yytext); return STARTMAIN;}
ENDMAIN     { yylval = strdup(yytext); return ENDMAIN;}
INTEGER     { yylval = strdup(yytext); return INTEGER;}
CHAR        { yylval = strdup(yytext); return CHAR;}
VARS        { yylval = strdup(yytext); return VARS;}
WHILE       { yylval = strdup(yytext); return WHILE;}
ENDWHILE    { yylval = strdup(yytext); return ENDWHILE;}
FOR     { yylval = strdup(yytext); return FOR;}
ENDFOR      { yylval = strdup(yytext); return ENDFOR;}
TO      { yylval = strdup(yytext); return TO;}
STEP        { yylval = strdup(yytext); return STEP;}
PRINT       { yylval = strdup(yytext); return PRINT;}

IF      { yylval = strdup(yytext); return IF;}
THEN        { yylval = strdup(yytext); return THEN;}
ELSE        { yylval = strdup(yytext); return ELSE;}
ELSEIF      { yylval = strdup(yytext); return ELSEIF;}
ENDIF       { yylval = strdup(yytext); return ENDIF;}
SWITCH      { yylval = strdup(yytext); return SWITCH;}
CASE        { yylval = strdup(yytext); return CASE;}
DEFAULT     { yylval = strdup(yytext); return DEFAULT;}
ENDSWITCH   { yylval = strdup(yytext); return ENDSWITCH;}
```

Αρχικά κάνουμε include τις απαραίτητες βιβλιοθήκες.

Με το noyywrap στην ουσία λέμε στο Flex να διαβάσει μόνο ένα input file. Με το yylineno το Flex φτιάχνει ένα scanner που καταχωρεί τον αριθμό στην σειρά της γραμμής που διαβάζεται κάθε φορά.

Δηλώνουμε τι είναι αριθμός,χαρακτήρας κτλ χρησιμοποιώντας regular expressions για να καλύψουμε οποιαδήποτε περίπτωση μπορεί να δημιουργηθεί.

Δήλωση δεσμευμένων λέξεων.

Η yylval είναι μια global μεταβλητή που χρησιμοποιούμε για να περάσουμε ένα sematic value που έχει σχέση με ένα token από το lexer στο parser. Για να μην γίνει κάποιο memory leak με το strdup(yytext) χρησιμοποιούμε την yylval.

```
STRUCT      { yylval = strdup(yytext); return STRUCT;}
ENDSTRUCT   { yylval = strdup(yytext); return ENDSTRUCT;}
TYPEDEF     { yylval = strdup(yytext); return TYPEDEF;}



AND     { yylval = strdup(yytext); return AND;}
OR      { yylval = strdup(yytext); return OR;}

{num}       { yylval = strdup(yytext); return INT; }
{word}      { yylval = strdup(yytext); return CH;}

"+"     { return '+'; }
"-"     { return '-'; }
"*"     { return '*'; }
"/"     { return '/'; }
"^"     { return '^'; }
"="     { return ASSIGN_OP;}
","     { return CM;  }
";"     { return SC;  }
">"     { return GT;  }
"<"     { return LT;  }
"=="        { return IS_EQ;  }
"!="        { return ISNOT_EQ;  }
\".*?\"     { yylval = strdup(yytext); return QT;  }

\n          { yylval = strdup(yytext);return NL;}
[' '\t\r] { }
"("         { yylval = strdup(yytext);return LP;}
")"         { yylval = strdup(yytext);return RP;}
"["         { yylval = strdup(yytext);return LB;}
"]"         { yylval = strdup(yytext);return RB;}
":"         { yylval = strdup(yytext);return CC;}

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

<<EOF>>     {return EOF;}
.           ;

%%
```

Δήλωση δεσμευμένων λέξεων για το ερώτημα 2.

Πρέπει να μπορούμε να αναγνωρίζουμε πράξεις, συγκρίσεις, εντολές ανάθεσης, σύνθετες παραστάσεις, συνδυασμούς κυριολεκτικων μεταβλητων και παρενθέσεων , συναρτήσεις, ορισμοί πινάκων και ότι άλλο έχει ζητηθεί από την εκφώνηση.

Ερώτημα 4
Αναγνώριση single line comment και multiline comment

Αναγνωρίζουμε το EOF ως το end of file.

parser.y

```
%{
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "symbol.h"
#define YYSTYPE char*
void yyerror(char *s);
int open_par=0;
int err=0;
extern FILE *yyin;
extern FILE *yyout;
extern int yylineno;
void uinput(char *name)
{
  utypes *t;
  t=ucheck(name);
  if(t==0)
  {
    t=uinsert(name);
  }
  else{err+=1; yyerror("name already exists");}
}
void finput(char *name)
{
  ufunc *t;
  t=fcheck(name);
  if(t==0)
  {
    t=finsert(name);
  }
  else{err+=1; yyerror("function name does not exist");}
}
void vinput(char *name)
{
  uvar *t;
  t=vcheck(name);
  if(t==0)
  {
    t=vinsert(name);
  }
  else{err+=1; yyerror("variable already exists");}
}
%}
```

Ερώτημα 3:
Σε κάθε δήλωση γίνεται έλεγχος για ύπαρξη αντικειμένου με το ίδιο όνομα. Για μια μεταβλητη δεν μας απασχόλεί ο τύπος της. Στις συναρτήσεις κατά την κλήση, δεν ελέγχουμε αν καλείται σωστά, αλλά μόνο αν υπάρχει το όνομα.

```
%define api.value.type {char*}

%token INT
%token CH
%token WS
%token CM
%token SC
%token CC
%token PROGRAM
%token FUNCTION
%token END_FUNCTION
%token RETURN
%token ASSIGN_OP
%token STARTMAIN
%token ENDMAIN
%token VARS
%token INTEGER
%token CHAR
%token NL
%token RP
%token LP
%token RB
%token LB
%token VAR
%token WHILE
%token ENDWHILE
%token FOR
%token ENDFOR
%token TO
%token STEP
%token PRINT
%token IF
%token THEN
%token ELSE
%token ELSEIF
%token ENDIF

%token SWITCH
%token CASE
%token DEFAULT
%token ENDSWITCH

%token STRUCT
%token ENDSTRUCT
%token TYPEDEF
```

```
%token QT
%token AND
%token OR
%token GT
%token LT
%token IS_EQ
%token ISNOT_EQ


%left   '+' '-'
%left   '*' '/'
%right  '^'

%start program

%%

program: prog {if(err==0){fprintf(stdout,"%s\n",$1);fprintf(stdout, "\nSyntax Correc
t\n");}else{} };
prog: PROGRAM CH NL structs dec_functions main NL{$
$=strcat(strcat(strcat(strcat(strcat(strcat($1," "),$2),$3),$4),$5),$6);}
;

dec_functions:  %empty  {$$="";}
    |FUNCTION CH args NL body RETURN CH SC NL END_FUNCTION NL{finput($2);$
$=strcat(strcat(strcat(strcat(strcat(strcat(strcat(strcat(strcat(strcat(strcat(
$1," "),$2),$3),$4),$5),$6)," "),$7),";"),"\n"),$10),$11);vTable=(uvar *)0;}
    |FUNCTION CH args NL body RETURN INT SC NL END_FUNCTION NL{finput($2);$
$=strcat(strcat(strcat(strcat(strcat(strcat(strcat(strcat(strcat(strcat(strcat(
$1," "),$2),$3),$4),$5),$6)," "),$7),";"),"\n"),$10),$11);vTable=(uvar *)0;}
;

struct: %empty  {$$="";}
  |STRUCT CH NL dec_var ENDSTRUCT NL {uinput( $2 ); $
$=strcat(strcat(strcat(strcat(strcat(strcat($1," "),$2),$3),$4),$5),$6);vTable=(uvar *
)0; }
  |TYPEDEF STRUCT CH NL dec_var CH ENDSTRUCT NL {if(strcmp($3,$6)==0){uinput( $3 );$
$=strcat(strcat(strcat(strcat(strcat(strcat(strcat(strcat(strcat(strcat($1," "),$2),"
"),$3),$4),$5),$6)," "),$7),$8);} else {yyerror("Struct names dont match"); err+=1;} v
Table=(uvar *)0;}
;

structs: struct     {$$=$1;}
  |structs struct {$$=strcat($1,$2);}
;

body:    dec_var        {$$=$1;}
```

Ερώτημα 2

```
  |instructions       {$$=$1;}
  |body instructions  {$$=strcat($1,$2);}
;

args: LP vars RP        {$$=strcat(strcat($1,$2),$3);}
;
vars:     INTEGER CH        {$$=strcat(strcat($1," "),$2);}
  | CHAR CH       {$$=strcat(strcat($1," "),$2);}
  | vars CM INTEGER CH    {$$=strcat(strcat(strcat(strcat($1,","),$3)," "),$4);}
  | vars CM CHAR CH   {$$=strcat(strcat(strcat(strcat($1,","),$3)," "),$4);}
;

dec_var: VARS NL var_dec    {$$=strcat(strcat($1,"\n"),$3);}
;
var_dec: %empty {$$="";}
  |INTEGER CH SC NL               {vinput($2);$$=strcat(strcat(strcat($1," "),
$2),";\n");}
  |INTEGER CH SC                  {vinput($2);$$=strcat(strcat(strcat($1," "),
$2),";");}
  |INTEGER CH ASSIGN_OP INT SC NL         {vinput($2);$
$=strcat(strcat(strcat(strcat(strcat($1," "),$2),"="),$4),";\n");}
  |INTEGER CH ASSIGN_OP INT SC            {vinput($2);$
$=strcat(strcat(strcat(strcat(strcat($1," "),$2),"="),$4),";\n");}
  |INTEGER CH LB INT RB SC NL         {vinput($2);$
$=strcat(strcat(strcat(strcat(strcat($1," "),$2),"["),$4),"];\n");}
  |INTEGER CH LB INT RB SC            {vinput($2);$
$=strcat(strcat(strcat(strcat(strcat($1," "),$2),"["),$4),"];");}
  |CHAR CH SC NL                  {vinput($2);$$=strcat(strcat(strcat($1," "),
$2),";\n");}
  |CHAR CH SC                     {vinput($2);$$=strcat(strcat(strcat($1," "),
$2),";");}
  |CHAR CH LB INT RB SC NL            {vinput($2);$
$=strcat(strcat(strcat(strcat(strcat($1," "),$2),"["),$4),"];\n");}
  |CHAR CH LB INT RB SC               {vinput($2);$
$=strcat(strcat(strcat(strcat(strcat($1," "),$2),"["),$4),"];");}
  |CHAR CH ASSIGN_OP CH SC NL         {vinput($2);$
$=strcat(strcat(strcat(strcat(strcat($1," "),$2),"="),$4),";\n");}
  |CHAR CH ASSIGN_OP CH SC            {vinput($2);$
$=strcat(strcat(strcat(strcat(strcat($1," "),$2),"="),$4),";\n");}
  |CH CH SC NL                    {vinput($2);if(ucheck($1)==0){yyerror("No such data
type");err+=1;}$$=strcat(strcat(strcat($1," "),$2),";\n");}
  |CH CH SC                   {vinput($2);if(ucheck($1)==0){yyerror("No such data type
");err+=1;}$$=strcat(strcat(strcat($1," "),$2),";");}
  |var_dec CHAR CH SC NL              {vinput($3);$
$=strcat(strcat(strcat(strcat($1,$2)," "),$3),";\n");}
  |var_dec CHAR CH SC                 {vinput($3);$
$=strcat(strcat(strcat(strcat($1,$2)," "),$3),";");}
```

```
  |var_dec INTEGER CH SC NL           {vinput($3);$
$=strcat(strcat(strcat(strcat($1,$2)," "),$3),";\n");}
  |var_dec INTEGER CH SC              {vinput($3);$
$=strcat(strcat(strcat(strcat($1,$2)," "),$3),";");}
  |var_dec INTEGER CH LB INT RB SC NL     {vinput($3);$
$=strcat(strcat(strcat(strcat(strcat(strcat($1,$2)," "),$3),"["),$5),"];\n");}
  |var_dec INTEGER CH LB INT RB SC        {vinput($3);$
$=strcat(strcat(strcat(strcat(strcat(strcat($1,$2)," "),$3),"["),$5),"];");}
  |var_dec CHAR CH LB INT RB SC NL        {vinput($3);$
$=strcat(strcat(strcat(strcat(strcat(strcat($1,$2)," "),$3),"["),$5),"];\n");}
  |var_dec CHAR CH LB INT RB SC           {vinput($3);$
$=strcat(strcat(strcat(strcat(strcat(strcat($1,$2)," "),$3),"["),$5),"];");}
  |var_dec INTEGER CH ASSIGN_OP INT SC NL     {vinput($3);$
$=strcat(strcat(strcat(strcat(strcat(strcat($1,$2)," "),$3),"="),$5),";\n");}
  |var_dec INTEGER CH ASSIGN_OP INT SC        {vinput($3);$
$=strcat(strcat(strcat(strcat(strcat(strcat($1,$2)," "),$3),"="),$5),";");}
  |var_dec CHAR CH ASSIGN_OP CH SC NL     {vinput($3);$
$=strcat(strcat(strcat(strcat(strcat(strcat($1,$2)," "),$3),"="),$5),";\n");}
  |var_dec CHAR CH ASSIGN_OP CH SC        {vinput($3);$
$=strcat(strcat(strcat(strcat(strcat(strcat($1,$2)," "),$3),"="),$5),";");}
  |var_dec INTEGER CH ASSIGN_OP math SC NL    {vinput($3);$
$=strcat(strcat(strcat(strcat(strcat(strcat($1,$2)," "),$3),"="),$5),";\n");}
  |var_dec INTEGER CH ASSIGN_OP math SC       {vinput($3);$
$=strcat(strcat(strcat(strcat(strcat(strcat($1,$2)," "),$3),"="),$5),";");}
;

main: STARTMAIN NL body ENDMAIN {$$=strcat(strcat(strcat($1,"\n"),$3),$4);}
;

instructions:    instruction instruction    {$$=strcat($1,$2);}
    |instruction instructions   {$$=strcat($1,$2);}
;
instruction:    %empty              {$$="";}
    |CH ASSIGN_OP INT SC        {if(vcheck($1)==0){yyerror("No such variable
");err+=1;}$$=strcat(strcat(strcat($1,"="),$3),";");}
    |CH ASSIGN_OP CH SC         {if(vcheck($1)==0 || vcheck($3)==0){yyerror("No such v
ariable");err+=1;}$$=strcat(strcat(strcat($1,"="),$3),";");}
    |CH ASSIGN_OP QT SC         {if(vcheck($1)==0){yyerror("No such variable
");err+=1;}$$=strcat(strcat(strcat($1,"="),$3),";");}
    |CH ASSIGN_OP math      {if(vcheck($1)==0){yyerror("No such variable");err+=1;}$
$=strcat(strcat(strcat($1,"="),$3),"");}
    |CH ASSIGN_OP function SC   {if(vcheck($1)==0){yyerror("No such variable
");err+=1;}$$=strcat(strcat(strcat($1,"="),$3),";");}
    |function SC            {$$=strcat($1,";");}
    |while              {$$=$1;}
    |for                {$$=$1;}
    |print              {$$=$1;}
```

```
    |if              {$$=$1;}
    |switch            {$$=$1;}
    |NL              {$$=$1;}
;
function:  CH LP csv RP     {if(fcheck($1)==0){yyerror("No such function");err+=1;}$
$=strcat(strcat(strcat($1,$2),$3),$4);}
   |CH LP CH  RP     {if(fcheck($1)==0){yyerror("No such function");err+=1;}$
$=strcat(strcat(strcat($1,$2),$3),$4);}
   |CH LP INT RP     {if(fcheck($1)==0){yyerror("No such function");err+=1;}$
$=strcat(strcat(strcat($1,$2),$3),$4);}
;
csv:     CH CM CH       {if(vcheck($1)==0 || vcheck($3)==0){yyerror("No such variable
");err+=1;}$$=strcat(strcat($1,","),$3);}
  |INT CM INT     {$$=strcat(strcat($1,","),$3);}
  |CH CM INT      {if(vcheck($1)==0){yyerror("No such variable");err+=1;}$
$=strcat(strcat($1,","),$3);}
  |INT CM CH      {if(vcheck($3)==0){yyerror("No such variable");err+=1;}$
$=strcat(strcat($1,","),$3);}
  |function CM function   {$$=strcat(strcat($1,","),$3);}
  |CH CM function     {if(vcheck($1)==0){yyerror("No such variable");err+=1;}$
$=strcat(strcat($1,","),$3);}
  |function CM CH     {if(vcheck($3)==0){yyerror("No such variable");err+=1;}$
$=strcat(strcat($1,","),$3);}
  |INT CM function    {$$=strcat(strcat($1,","),$3);}
  |function CM INT    {$$=strcat(strcat($1,","),$3);}
  |csv CM CH      {if(vcheck($3)==0){yyerror("No such variable");err+=1;}$
$=strcat(strcat($1,","),$3);}
  |csv CM INT         {$$=strcat(strcat($1,","),$3);}
  |csv CM function    {$$=strcat(strcat($1,","),$3);}
;

math:    CH"+"CH        {if(vcheck($1)==0 || vcheck($3)==0){yyerror("No such variable
");err+=1;}$$=strcat(strcat($1,"+"),$3);}
  |CH"-"CH        {if(vcheck($1)==0 || vcheck($3)==0){yyerror("No such variable
");err+=1;}$$=strcat(strcat($1,"-"),$3);}
  |CH"*"CH        {if(vcheck($1)==0 || vcheck($3)==0){yyerror("No such variable
");err+=1;}$$=strcat(strcat($1,"*"),$3);}
  |CH"^"CH        {if(vcheck($1)==0 || vcheck($3)==0){yyerror("No such variable
");err+=1;}$$=strcat(strcat($1,"^"),$3);}
  |CH"/"CH        {if(vcheck($1)==0 || vcheck($3)==0){yyerror("No such variable
");err+=1;}$$=strcat(strcat($1,"/"),$3);}
  |INT"+"INT      {$$=strcat(strcat($1,"+"),$3);}
  |INT"-"INT      {$$=strcat(strcat($1,"-"),$3);}
  |INT"*"INT      {$$=strcat(strcat($1,"*"),$3);}
  |INT"^"INT      {$$=strcat(strcat($1,"^"),$3);}
  |INT"/"INT      {$$=strcat(strcat($1,"/"),$3);}
  |function"+"function    {$$=strcat(strcat($1,"+"),$3);}
```

```
  |function"-"function    {$$=strcat(strcat($1,"-"),$3);}
  |function"*"function    {$$=strcat(strcat($1,"*"),$3);}
  |function"^"function    {$$=strcat(strcat($1,"^"),$3);}
  |function"/"function    {$$=strcat(strcat($1,"/"),$3);}
  |function"+"INT     {$$=strcat(strcat($1,"+"),$3);}
  |function"-"INT     {$$=strcat(strcat($1,"-"),$3);}
  |function"*"INT     {$$=strcat(strcat($1,"*"),$3);}
  |function"^"INT     {$$=strcat(strcat($1,"^"),$3);}
  |function"/"INT     {$$=strcat(strcat($1,"/"),$3);}
  |function"+"CH      {if(vcheck($3)==0){yyerror("No such variable");err+=1;}$
$=strcat(strcat($1,"+"),$3);}
  |function"-"CH      {if(vcheck($3)==0){yyerror("No such variable");err+=1;}$
$=strcat(strcat($1,"-"),$3);}
  |function"*"CH      {if(vcheck($3)==0){yyerror("No such variable");err+=1;}$
$=strcat(strcat($1,"*"),$3);}
  |function"^"CH      {if(vcheck($3)==0){yyerror("No such variable");err+=1;}$
$=strcat(strcat($1,"^"),$3);}
  |function"/"CH      {if(vcheck($3)==0){yyerror("No such variable");err+=1;}$
$=strcat(strcat($1,"/"),$3);}
  |CH"+"INT       {if(vcheck($1)==0){yyerror("No such variable");err+=1;}$
$=strcat(strcat($1,"+"),$3);}
  |CH"-"INT       {if(vcheck($1)==0){yyerror("No such variable");err+=1;}$
$=strcat(strcat($1,"-"),$3);}
  |CH"*"INT       {if(vcheck($1)==0){yyerror("No such variable");err+=1;}$
$=strcat(strcat($1,"*"),$3);}
  |CH"^"INT       {if(vcheck($1)==0){yyerror("No such variable");err+=1;}$
$=strcat(strcat($1,"^"),$3);}
  |CH"/"INT       {if(vcheck($1)==0){yyerror("No such variable");err+=1;}$
$=strcat(strcat($1,"/"),$3);}
  |CH"+"function      {if(vcheck($1)==0){yyerror("No such variable");err+=1;}$
$=strcat(strcat($1,"+"),$3);}
  |CH"-"function      {if(vcheck($1)==0){yyerror("No such variable");err+=1;}$
$=strcat(strcat($1,"-"),$3);}
  |CH"*"function      {if(vcheck($1)==0){yyerror("No such variable");err+=1;}$
$=strcat(strcat($1,"*"),$3);}
  |CH"^"function      {if(vcheck($1)==0){yyerror("No such variable");err+=1;}$
$=strcat(strcat($1,"^"),$3);}
  |CH"/"function      {if(vcheck($1)==0){yyerror("No such variable");err+=1;}$
$=strcat(strcat($1,"/"),$3);}
  |INT"+"CH       {if(vcheck($3)==0){yyerror("No such variable");err+=1;}$
$=strcat(strcat($1,"+"),$3);}
  |INT"-"CH       {if(vcheck($3)==0){yyerror("No such variable");err+=1;}$
$=strcat(strcat($1,"-"),$3);}
  |INT"*"CH       {if(vcheck($3)==0){yyerror("No such variable");err+=1;}$
$=strcat(strcat($1,"*"),$3);}
  |INT"^"CH       {if(vcheck($3)==0){yyerror("No such variable");err+=1;}$
$=strcat(strcat($1,"^"),$3);}
```

```
  |INT'/'CH       {if(vcheck($3)==0){yyerror("No such variable");err+=1;}$
$=strcat(strcat($1,"/"),$3);}
  |INT'+'function     {$$=strcat(strcat($1,"+"),$3);}
  |INT'-'function     {$$=strcat(strcat($1,"-"),$3);}
  |INT'*'function     {$$=strcat(strcat($1,"*"),$3);}
  |INT'^'function     {$$=strcat(strcat($1,"^"),$3);}
  |INT'/'function     {$$=strcat(strcat($1,"/"),$3);}
  |math'+'CH      {if(vcheck($3)==0){yyerror("No such variable");err+=1;}$
$=strcat(strcat($1,"+"),$3);}
  |math'-'CH      {if(vcheck($3)==0){yyerror("No such variable");err+=1;}$
$=strcat(strcat($1,"-"),$3);}
  |math'*'CH      {if(vcheck($3)==0){yyerror("No such variable");err+=1;}$
$=strcat(strcat($1,"*"),$3);}
  |math'^'CH      {if(vcheck($3)==0){yyerror("No such variable");err+=1;}$
$=strcat(strcat($1,"^"),$3);}
  |math'/'CH      {if(vcheck($3)==0){yyerror("No such variable");err+=1;}$
$=strcat(strcat($1,"/"),$3);}
  |math'+'INT     {$$=strcat(strcat($1,"+"),$3);}
  |math'-'INT     {$$=strcat(strcat($1,"-"),$3);}
  |math'*'INT     {$$=strcat(strcat($1,"*"),$3);}
  |math'^'INT     {$$=strcat(strcat($1,"^"),$3);}
  |math'/'INT     {$$=strcat(strcat($1,"/"),$3);}
  |math'+'function    {$$=strcat(strcat($1,"+"),$3);}
  |math'-'function    {$$=strcat(strcat($1,"-"),$3);}
  |math'*'function    {$$=strcat(strcat($1,"*"),$3);}
  |math'^'function    {$$=strcat(strcat($1,"^"),$3);}
  |math'/'function    {$$=strcat(strcat($1,"/"),$3);}
  |math SC NL     {$$=strcat($1,";\n");}
  |math SC        {$$=strcat($1,";");}
;
comp:    CH GT CH       {if(vcheck($1)==0 || vcheck($3)==0){yyerror("No such variable
");err+=1;}$$=strcat(strcat($1,">"),$3);}
  |CH GT INT      {if(vcheck($1)==0){yyerror("No such variable");err+=1;}$
$=strcat(strcat($1,">"),$3);}
  |CH GT function     {if(vcheck($1)==0){yyerror("No such variable");err+=1;}$
$=strcat(strcat($1,">"),$3);}
  |CH GT LP math RP   {if(vcheck($1)==0){yyerror("No such variable");err+=1;}$
$=strcat(strcat(strcat(strcat($1,">"),$3),$4),$5);}
  |INT GT INT     {$$=strcat(strcat($1,">"),$3);}
  |INT GT CH      {if(vcheck($3)==0){yyerror("No such variable");err+=1;}$
$=strcat(strcat($1,">"),$3);}
  |INT GT function    {$$=strcat(strcat($1,">"),$3);}
  |INT GT LP math RP  {$$=strcat(strcat(strcat(strcat($1,">"),$3),$4),$5);}
  |function GT function   {$$=strcat(strcat($1,">"),$3);}
  |function GT CH     {if(vcheck($3)==0){yyerror("No such variable");err+=1;}$
$=strcat(strcat($1,">"),$3);}
  |function GT INT    {$$=strcat(strcat($1,">"),$3);}
```

```
  |function GT LP math RP {$$=strcat(strcat(strcat(strcat($1,">"),$3),$4),$5);}
  |CH LT CH       {if(vcheck($1)==0 || vcheck($3)==0){yyerror("No such variable
");err+=1;}$$=strcat(strcat($1,"<"),$3);}
  |CH LT INT      {if(vcheck($1)==0){yyerror("No such variable");err+=1;}$
$=strcat(strcat($1,"<"),$3);}
  |CH LT function     {if(vcheck($1)==0){yyerror("No such variable");err+=1;}$
$=strcat(strcat($1,"<"),$3);}
  |CH LT LP math RP   {if(vcheck($1)==0){yyerror("No such variable");err+=1;}$
$=strcat(strcat(strcat(strcat($1,"<"),$3),$4),$5);}
  |INT LT INT     {$$=strcat(strcat($1,"<"),$3);}
  |INT LT CH      {if(vcheck($3)==0){yyerror("No such variable");err+=1;}$
$=strcat(strcat($1,"<"),$3);}
  |INT LT function    {$$=strcat(strcat($1,"<"),$3);}
  |INT LT LP math RP  {$$=strcat(strcat(strcat(strcat($1,"<"),$3),$4),$5);}
  |function LT function   {$$=strcat(strcat($1,"<"),$3);}
  |function LT CH     {if(vcheck($3)==0){yyerror("No such variable");err+=1;}$
$=strcat(strcat($1,"<"),$3);}
  |function LT INT    {$$=strcat(strcat($1,"<"),$3);}
  |function LT LP math RP {$$=strcat(strcat(strcat(strcat($1,"<"),$3),$4),$5);}
  |CH IS_EQ CH           {if(vcheck($1)==0 || vcheck($3)==0){yyerror("No such variabl
e");err+=1;}$$=strcat(strcat($1,"=="),$3);}
  |CH IS_EQ INT          {if(vcheck($1)==0){yyerror("No such variable");err+=1;}$
$=strcat(strcat($1,"=="),$3);}
  |CH IS_EQ function      {if(vcheck($1)==0){yyerror("No such variable");err+=1;}$
$=strcat(strcat($1,"=="),$3);}
  |CH IS_EQ LP math RP        {if(vcheck($1)==0){yyerror("No such variable");err+=1;}$
$=strcat(strcat(strcat(strcat($1,"=="),$3),$4),$5);}
  |INT IS_EQ INT          {$$=strcat(strcat($1,"=="),$3);}
  |INT IS_EQ CH           {if(vcheck($3)==0){yyerror("No such variable");err+=1;}$
$=strcat(strcat($1,"=="),$3);}
  |INT IS_EQ function     {$$=strcat(strcat($1,"=="),$3);}
  |INT IS_EQ LP math RP       {$$=strcat(strcat(strcat(strcat($1,"=="),$3),$4),$5);}
  |function IS_EQ function    {$$=strcat(strcat($1,"=="),$3);}
  |function IS_EQ CH      {if(vcheck($3)==0){yyerror("No such variable");err+=1;}$
$=strcat(strcat($1,"=="),$3);}
  |function IS_EQ INT     {$$=strcat(strcat($1,"=="),$3);}
  |function IS_EQ LP math RP  {$$=strcat(strcat(strcat(strcat($1,"=="),$3),$4),$5);}
  |CH ISNOT_EQ CH         {if(vcheck($1)==0 || vcheck($3)==0){yyerror("No such variabl
e");err+=1;}$$=strcat(strcat($1,"!="),$3);}
  |CH ISNOT_EQ INT        {if(vcheck($1)==0){yyerror("No such variable");err+=1;}$
$=strcat(strcat($1,"!="),$3);}
  |CH ISNOT_EQ function       {if(vcheck($1)==0){yyerror("No such variable");err+=1;}$
$=strcat(strcat($1,"!="),$3);}
  |CH ISNOT_EQ LP math RP     {if(vcheck($1)==0){yyerror("No such variable");err+=1;}$
$=strcat(strcat(strcat(strcat($1,"!="),$3),$4),$5);}
  |INT ISNOT_EQ INT       {$$=strcat(strcat($1,"!="),$3);}
```

```
  |INT ISNOT_EQ CH        {if(vcheck($3)==0){yyerror("No such variable");err+=1;}$
$=strcat(strcat($1,"!="),$3);}
  |INT ISNOT_EQ function      {$$=strcat(strcat($1,"!="),$3);}
  |INT ISNOT_EQ LP math RP    {$$=strcat(strcat(strcat(strcat($1,"!="),$3),$4),$5);}
  |function ISNOT_EQ function {$$=strcat(strcat($1,"!="),$3);}
  |function ISNOT_EQ CH       {if(vcheck($3)==0){yyerror("No such variable");err+=1;}$
$=strcat(strcat($1,"!="),$3);}
  |function ISNOT_EQ INT      {$$=strcat(strcat($1,"!="),$3);}
  |function ISNOT_EQ LP math RP   {$$=strcat(strcat(strcat(strcat($1,"!="),$3),$4),
$5);}
  |comp AND comp          {$$=strcat(strcat(strcat(strcat($1," "),$2)," "),$3);}
  |comp OR comp           {$$=strcat(strcat(strcat(strcat($1," "),$2)," "),$3);}
;

while: WHILE LP comp RP NL body ENDWHILE NL {$
$=strcat(strcat(strcat(strcat(strcat(strcat(strcat($1,"("),$3),")"),$5),$6),$7),$8);}
;

for: FOR CH ASSIGN_OP INT TO INT STEP INT body ENDFOR {$
$=strcat(strcat(strcat(strcat(strcat(strcat(strcat(strcat(strcat(strcat(strcat(
strcat(strcat($1," "),$2),"="),$4)," "),$5)," "),$6)," "),$7)," "),$8),$9),$10);}
;

print:   PRINT LP QT RP SC      {$$=strcat(strcat(strcat(strcat($1,$2),$3),$4),";");}
  |PRINT LP QT CM csv RP SC   {$$=strcat(strcat(strcat(strcat(strcat($1,$2),$3),","),
$5),");");}
  |PRINT LP QT CM CH RP SC    {$$=strcat(strcat(strcat(strcat(strcat($1,$2),$3),","),
$5),");");}
  |PRINT LP QT CM INT RP SC   {$$=strcat(strcat(strcat(strcat(strcat($1,$2),$3),","),
$5),");");}
  |PRINT LP QT CM function RP SC  {$$=strcat(strcat(strcat(strcat(strcat($1,$2),
$3),","),$5),");");}
;


elseif:  ELSEIF LP comp RP THEN         {$$=strcat(strcat(strcat(strcat($1,$2),$3),
$4),$5);}
  |elseif body ELSEIF LP comp RP THEN {$
$=strcat(strcat(strcat(strcat(strcat(strcat($1,$2),$3),$4),$5),$6),$7);}
;

if:  IF LP comp RP THEN body ENDIF          {$
$=strcat(strcat(strcat(strcat(strcat(strcat($1,$2),$3),$4),$5),$6),$7);}
  |IF LP comp RP THEN body elseif body ENDIF  {$
$=strcat(strcat(strcat(strcat(strcat(strcat(strcat(strcat($1,$2),$3),$4),$5),$6),$7),
$8),$9);}
```

```
    |IF LP comp RP THEN body ELSE   body ENDIF  {$
$=strcat(strcat(strcat(strcat(strcat(strcat(strcat(strcat($1,$2),$3),$4),$5),$6),$7),
$8),$9);}
  |IF LP comp RP THEN body elseif body ELSE body ENDIF    {$
$=strcat(strcat(strcat(strcat(strcat(strcat(strcat(strcat(strcat(strcat($1,$2),$3),
$4),$5),$6),$7),$8),$9),$10),$11);}
;

expr:    INT        {$$=$1;}
  |CH         {if(vcheck($1)==0){yyerror("No such variable");err+=1;}$$=$1;}
  |function   {$$=$1;}
  |math       {$$=$1;}
;

switch:  SWITCH LP expr RP NL case ENDSWITCH                {$
$=strcat(strcat(strcat(strcat(strcat(strcat($1,$2),$3),$4),$5),$6),$7);}
  |SWITCH LP expr RP NL case DEFAULT instructions ENDSWITCH   {$
$=strcat(strcat(strcat(strcat(strcat(strcat(strcat($1,$2),$3),$4),$5),$6),$7),
$8),$9);}
;

case:    CASE LP expr RP CC NL instructions     {$
$=strcat(strcat(strcat(strcat(strcat(strcat($1,$2),$3),$4),$5),$6),$7);}
  |case CASE LP expr RP CC NL instructions    {$
$=strcat(strcat(strcat(strcat(strcat(strcat(strcat($1,$2),$3),$4),$5),$6),$7),$8);}
;

%%

void yyerror(char *s)
{
  fprintf(stderr, "line:%d %s\n",yylineno, s);
}


int main ( int argc, char **argv  )
 {
 ++argv; --argc;
 if ( argc > 0 )
    yyin = fopen( argv[0], "r" );
 else
    yyin = stdin;
 yyout = fopen ( "outputtst", "w" );
 yyparse ();
 return 0;
 }
```

symbol.h

```
typedef struct utypes
{
  char *name;
  struct utypes *nptr;
}utypes;

utypes *uTable=(utypes *)0;
utypes *uinsert();
utypes *ucheck();


utypes * uinsert( char *name)
{
  utypes *ptr;
  ptr=(utypes *)malloc(sizeof(utypes));
  ptr->name=(char *)malloc(strlen(name)+1);
  strcpy(ptr->name,name);
  ptr->nptr=(struct utypes *)uTable;
  uTable=ptr;
  return ptr;
}

utypes * ucheck(char *name)
{
  utypes *ptr;
  for (ptr=uTable;ptr!= (utypes *)0;ptr=(utypes *)ptr->nptr)
  {
    if(strcmp(name,ptr->name)==0)
    {
      return ptr;
    }
  }

  return 0;
}

typedef struct ufunc
{
  char *name;
  struct ufunc *nptr;
}ufunc;

ufunc *fTable=(ufunc *)0;
ufunc *finsert();
ufunc *fcheck();
```

Ερώτημα 3
Τσεκάρουμε τις συναρτήσεις και τις μεταβλητές.

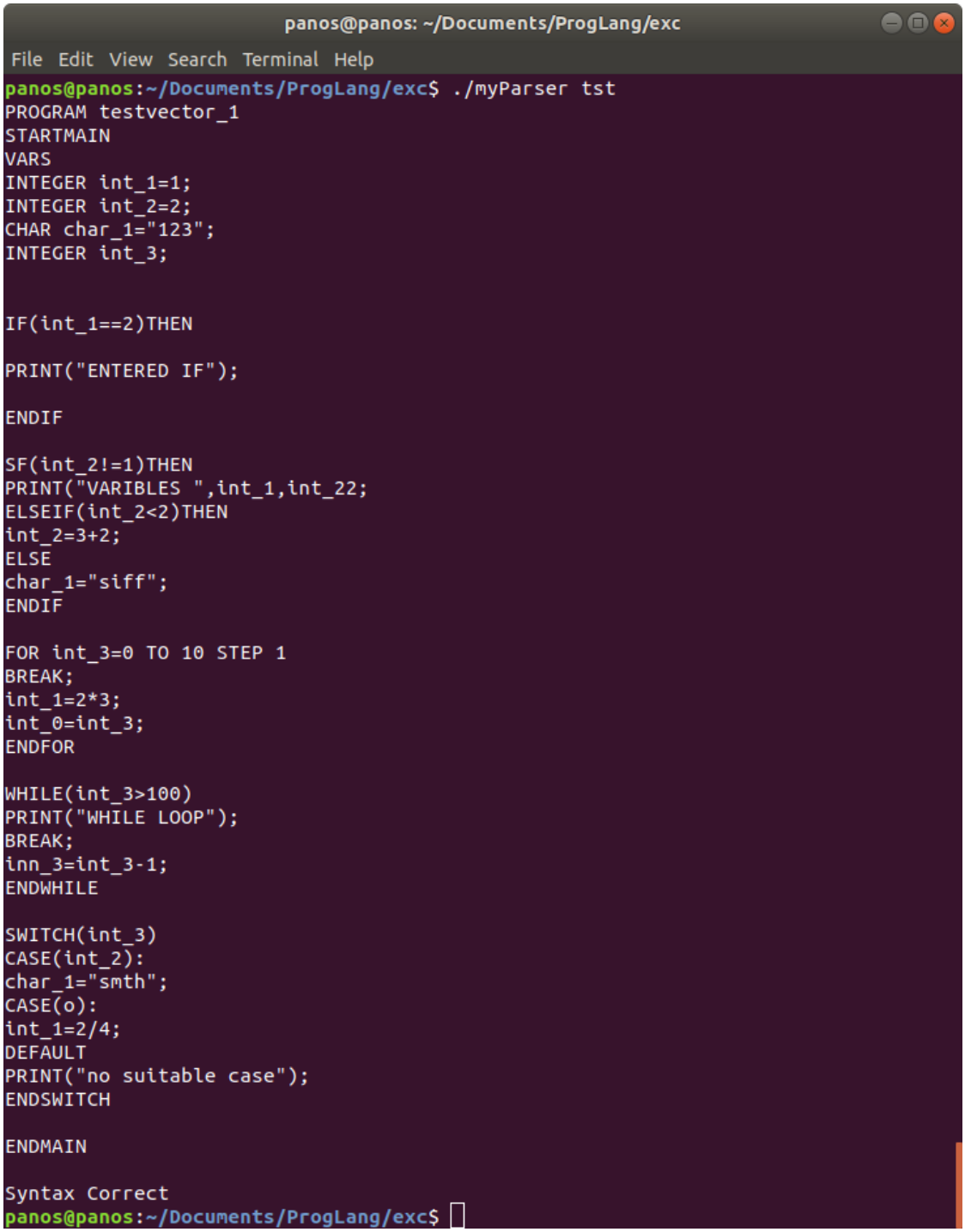
```
ufunc * finsert( char *name)
{
  ufunc *ptr;
  ptr=(ufunc *)malloc(sizeof(ufunc));
  ptr->name=(char *)malloc(strlen(name)+1);
  strcpy(ptr->name,name);
  ptr->nptr=(struct ufunc *)uTable;
  fTable=ptr;
  return ptr;
}

ufunc * fcheck(char *name)
{
  ufunc *ptr;
  for (ptr=fTable;ptr!= (ufunc *)0;ptr=(ufunc *)ptr->nptr)
  {
    if(strcmp(name,ptr->name)==0)
    {
      return ptr;
    }
  }

  return 0;
}

typedef struct uvar
{
  char *name;
  struct uvar *nptr;
}uvar;

uvar *vTable=(uvar *)0;
uvar *vinsert();
uvar *vcheck();


uvar * vinsert( char *name)
{
  uvar *ptr;
  ptr=(uvar *)malloc(sizeof(uvar));
  ptr->name=(char *)malloc(strlen(name)+1);
  strcpy(ptr->name,name);
  ptr->nptr=(struct uvar *)vTable;
  vTable=ptr;
  return ptr;
```

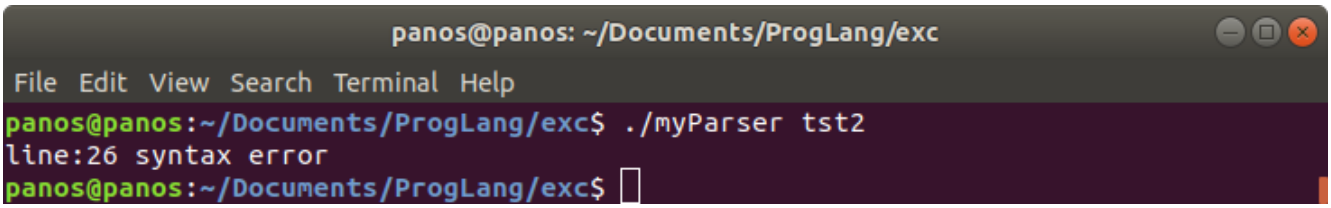
```
}

uvar * vcheck(char *name)
{
  uvar *ptr;
  for (ptr=vTable;ptr!= (uvar *)0;ptr=(uvar *)ptr->nptr)
  {
    if(strcmp(name,ptr->name)==0)
    {
      return ptr;
    }
  }

  return 0;
}
```

Assumptions - Examples

Question 1.b: When our program is called from the command line like this:
prompt>myParser.exe file.c
returns the program itself, choosing an implementation that ignores comments, and a diagnostic message about whether it is well-written or the appropriate error message

Example of a correctly written program





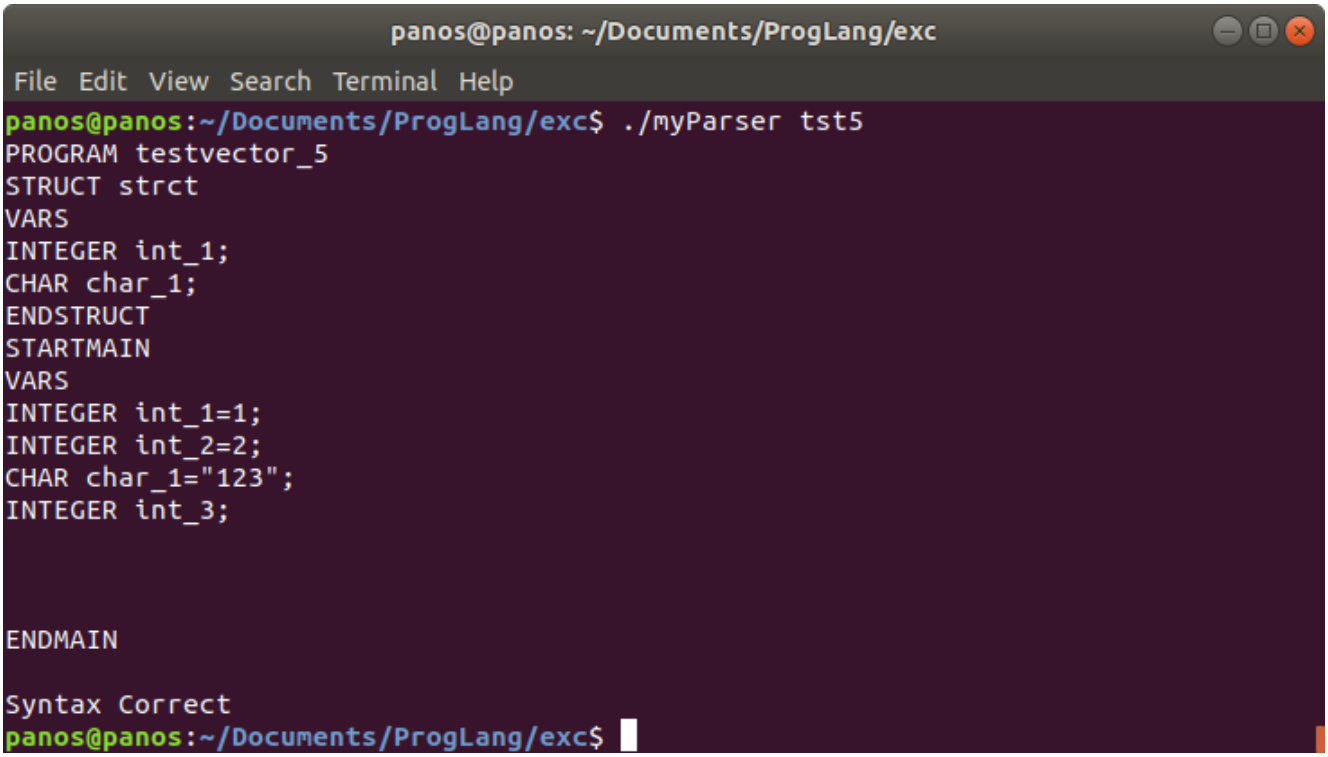
Example with a syntax error

When a syntax error appears it stops reading and informs where the error was found.

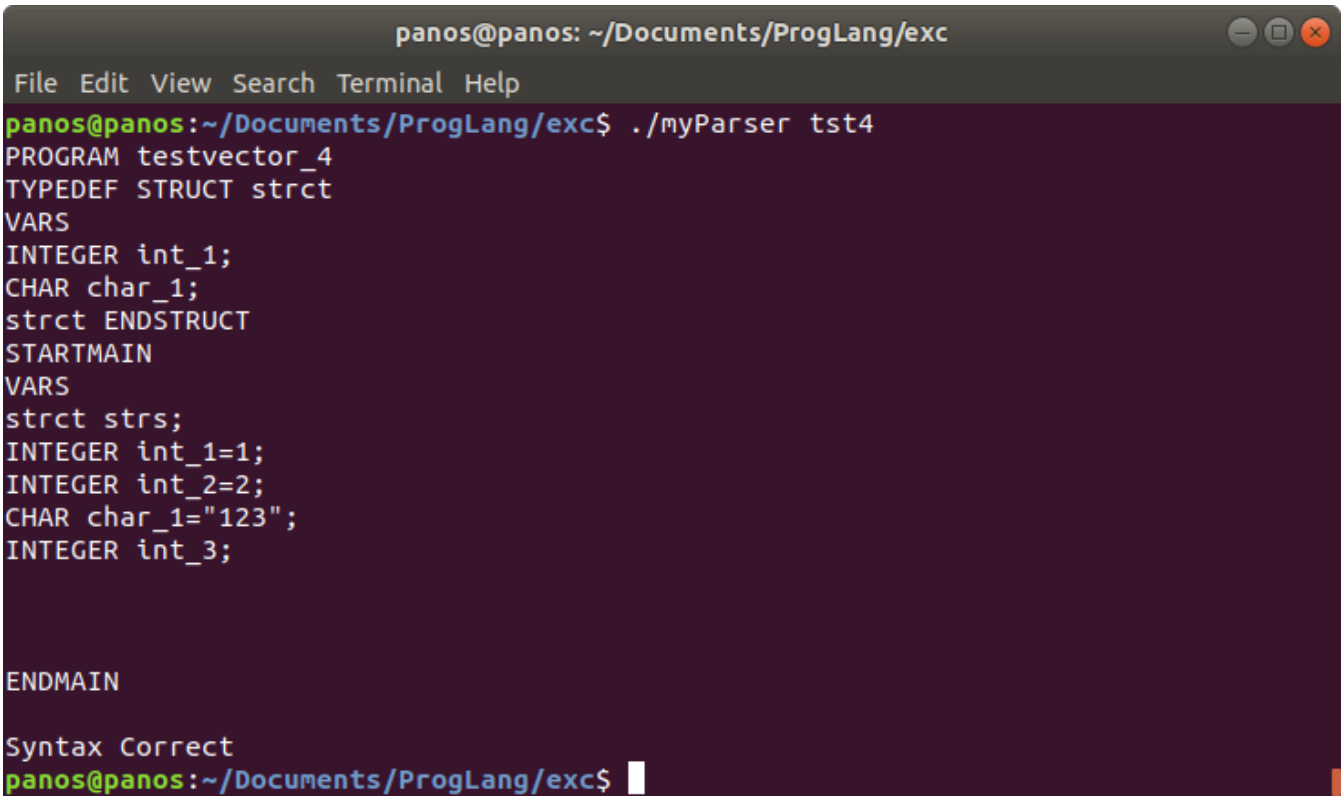
Question 2

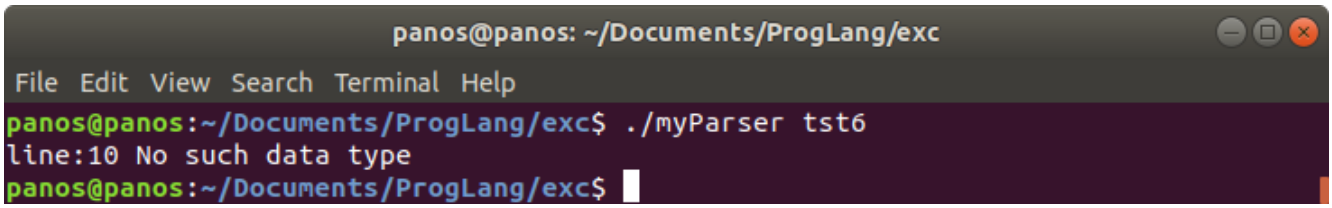
We've extended our parser by adding the ability to recognize any type name.

Example for TYPEDEF and using the type. Correct syntax.



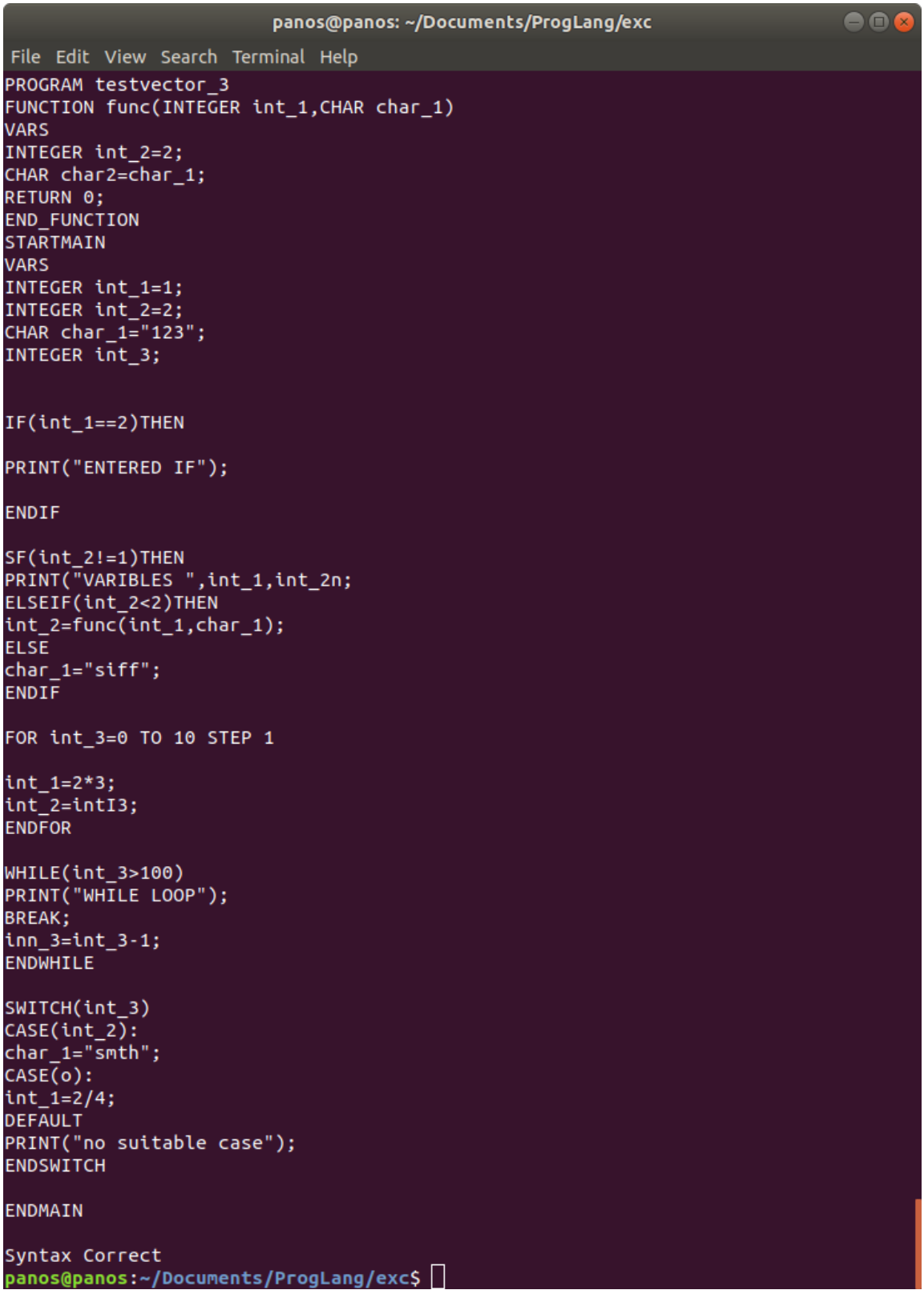
Example for simple STRUCT. Correct syntax.





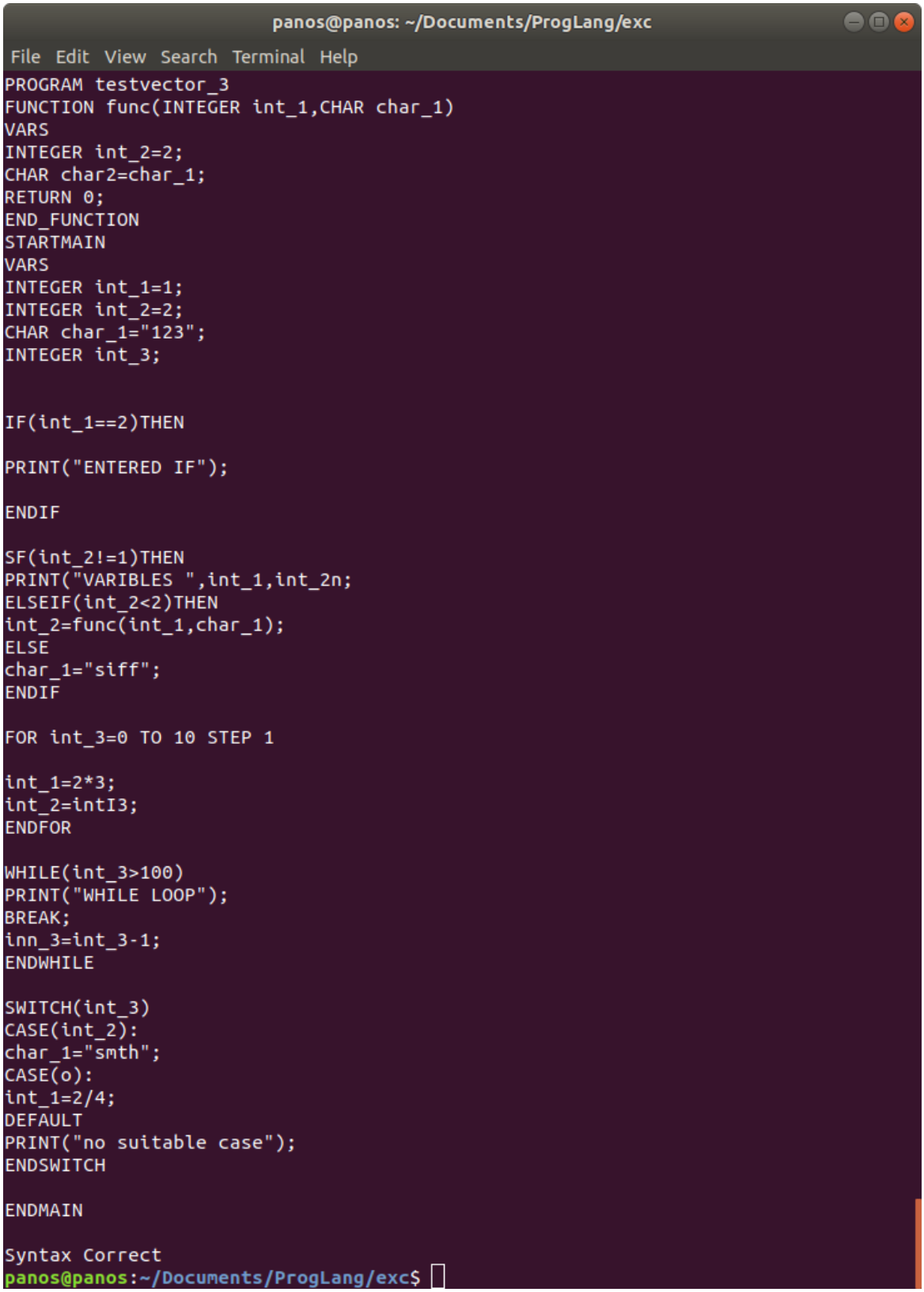
Example of using an undefined type.

Question 2

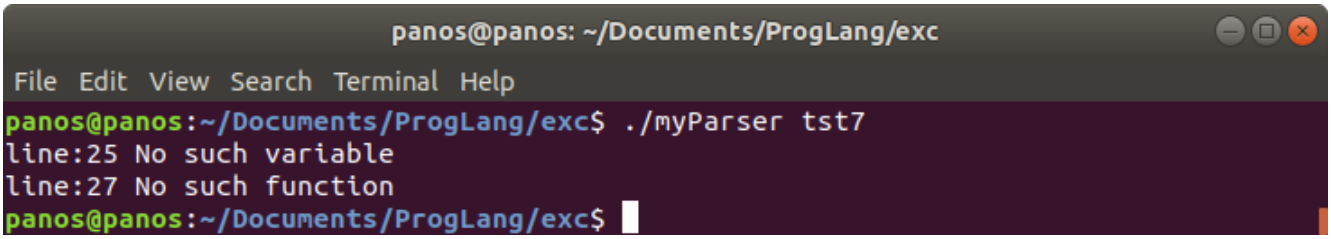


Example: Check for correct declaration of variables used in the program. Correct syntax

Function example.



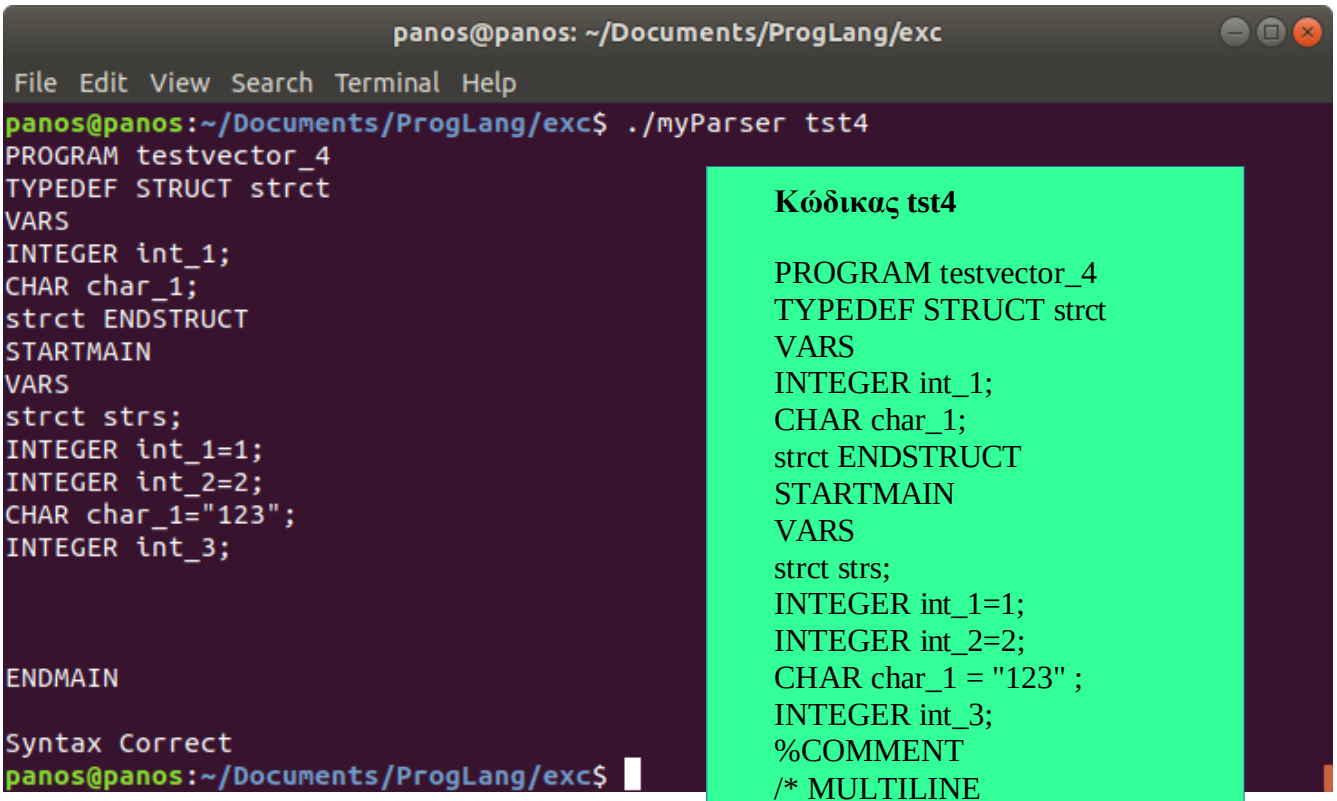
Example using non-existent variable and non-existent function.
Abort the parsing process with a function not found error.



Question 4

Comment support example.

We chose to implement both single-line and multi-line comments as required by the exercise.



Κώδικας tst4

```
PROGRAM testvector_4
TYPEDEF STRUCT strct
VARS
INTEGER int_1;
CHAR char_1;
strct ENDSTRUCT
STARTMAIN
VARS
strct strs;
INTEGER int_1=1;
INTEGER int_2=2;
CHAR char_1 = "123" ;
INTEGER int_3;
%COMMENT
/* MULTILINE
COMMENT
*/

ENDMAIN
```

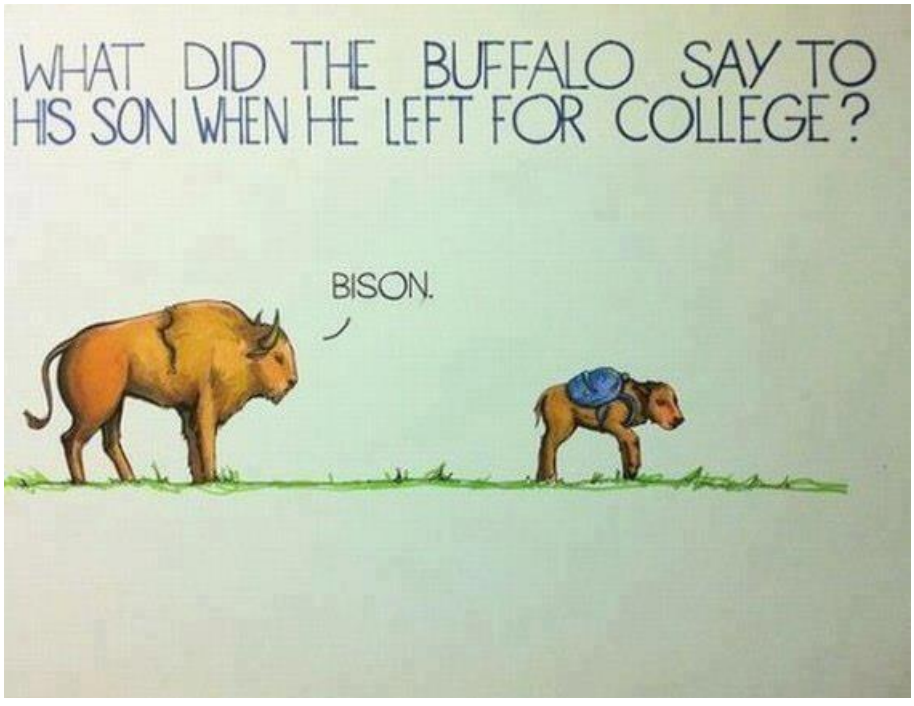
Comments

In the work, the implementation of a tool was requested which will simply check the syntactic and lexical correctness of a program. For this as an action on all rules it is a simple formatting of the rule to pass from the unchecked program to the terminal if it is correct otherwise an error is displayed. For some special errors we have assigned specific messages which, however, do not stop the reading of the unchecked program. Syntax errors detected by bison will block reading and simply display the syntax error message and the line number where it was found. When checking a user-defined function call, it is not syntax-checked that the correct arguments are given, only if a function with that name exists.

Finally, while writing the reference, we realized that we neglected to implement BREAK, which we added to the lexer and parser, but we didn't manage to incorporate the changes into the reference (that is, add it to the listing part of the code).

Bibliography

Slides Principles of Programming Languages and Translators. Bison manual by Charles Donnelly and Richard Stallman
Lexical Analysis with Flex by Vern Paxson, Will Estes and John Millaway flex & bison – John R. Levine wckdawe – compilers


WHAT DID THE BUFFALO SAY TO HIS SON WHEN HE LEFT FOR COLLEGE?
BISON.