Principles of Programming & Translator Languages

1)Description of BNF Grammar

First, we define the format of our code through the code. We first put the reserved word PROGRAM, then Args through which we define the name of the program, function_definition which is the grammar for defining functions, and main_definition in which we define the form of main.

Function_definition:

It starts with FUNCTION, then through name the name of the function is defined, then in parameters the parameters of the function are defined. Then we have line change and definition of the variables. Then the commands follow. We have the reserved words RETURN and END_FUNCTION.

Main_definition:

It starts with the reserved word STARTMAIN and continues with variable definition and the included commands. It ends with the reserved word ENDMAIN.

Commands:

The commands include assign, loop, check, print, break, comments.

Assign:

The grammar of variables using an assign operator which is the equal (=) then combined with a complex expression. The grammar of complex expression consists of expressions of addition or subtraction or multiplication or division or exponentiation, with or without parentheses.

Break:

Αποτελείται από την δεσμευμένη λέξη BREAK.

Print:

It consists of the reserved word print, parentheses inside which are constants, names, args, which are the strings printed by print, next to which the variables are called.

Comments:

The comment grammar of the same line consists of the % symbol and the value grammar is used to select the comments that the user wants to define.

Loop:

It consists of 2 subroutines, while and for.

While:

It starts with the reserved word while, and the condition is defined in parentheses. The grammar of the condition consists of 2 expressions between which there is a logical operator. After the condition there is a line change, the commands within the for are called, changing line by line. It ends with the reserved word ENDWHILE.

For:

It starts with the reserved word for. The bound word counter which is equal to a number from the grammar constant. We have the bound words to, the step in which we define the frequency at which our for will be executed. Next, we have a line break. Then the commands are called, and it ends with the bound word ENDFOR.

Check:

The grammar check consists of 2 subgrams, check1 and check2. The first contains if, while the second contains switch.

If:

It starts with the reserved word if. Inside parentheses is the condition under which it will be executed. Then with the reserved word THEN it continues with the commands. Inside parentheses and using the Kleene star we define the else contingencies of if. We have the reserved word else if, and next to it inside parentheses the condition, and then again commands follow. Then follows the reserved word else with the corresponding commands it contains. The if ends with the reserved word END IF.

Switch:

It starts with the word SWITCH, followed by the word CASE (complex expression) in parentheses, followed by the word CASE (complex expression), the colon symbol and the commands called. This is followed by the reserved word DEFAULT with a colon followed by commands. The grammar ends with the reserved word ENDSWITCH.

Struct:

It starts with the reserved word STRUCT followed by the type name of the struct. We have line change and variable definition as described below. The grammar closes with the reserved word ENDSTRUCT. Alternatively, this grammar can be used with the word TYPEDEF before the word STRUCT and the type name before the word ENDSTRUCT.

Μεταβλητές:

<var-declare>::=VARS <variables>*<semicolon> Ορισμός μεταβλητών

<variables>::=<characters> Μεταβλητές

|<integers>

<characters>::=CHAR<name>+ Χαρακτήρες

<integers>::=INTEGER<name>+ Ακέραιοι

<name>::= <Args><constant>*<coma>*

|<ARRAY>*<coma>*

<ARRAY>::=<Args><constant>* <'['constant']'> Πίνακες

Μικρές γραμματικές που βοήθησαν στην εκτέλεση των παραπάνω:

<Args>::=[a-zA-Z] Ορισμός strings

<coma>::=","

<constant>::=[0-9]

<semicolon>::=";"

<value>::= <name>

| <constant>

| <Args>

<logexp>::= >|<|==|!=|AND|OR Λογικοί τελεστές

<sinthiki>::= <value> <logexo> <value> Ορισμός λογικής έκφρασης

<mul-operators>::= "^"|"*"|"/" Αριθμητοί τελεστές

<ad-operators>::="+"|"-" Αριθμητοί τελεστές

<assign-operator>::="="

<parameters>::=<Args><constant>*

Lexical parser (flex)::

Syntactic analyser (bison):

Inputs flex&bison

1η Input:

```
PROGRAM code
STARTMAIN
VARS
INTEGER num,var;
SWITCH(var+1)
CASE(1)
PRINT("caseone"[,num]);
CASE(2)
PRINT("casetwo"[,num]);
DEFAULT
PRINT("casethree"[,num]);
```

ENDSWITCH

ENDMAIN

The above input uses the grammar of:

Code:

```
code: PRGM name NEWLINE struct function mainDefiniiton;
```

Main:

```
//main syntax
mainDefiniiton: SMN NEWLINE declarationstar NEWLINE prog_commands EMN
              ;
```

Declarationstar:

```
declarationstar: |declaration
               ;
```

Declaration:

```
declaration: VAR NEWLINE variables  SEMI;  //variable declaration syntax
```

Variables:

```
//variables syntax
variables:  variables variables
          |integers
          |characters
          ;
```

Integer & characters:

```
characters: ABC  name   //char syntax
          |characters array
          ;
integers: INT  name     //integer syntax
          |integers array
          ;
```

Name:

```
name: |name COMA str con     //name syntax (var,var1 ...)
    |name COMA str
    |str con
    |str
    ;
```

Array:

```
array: LAGY  con  RAGY ; //array syntax
```

Prog_commands:

```
//in commands commands syntax
prog_commands:  |prog_commands expr NEWLINE
                |prog_commands commands
                |expr NEWLINE
                |commands
                ;
```

Commands:

```
commands: commands commands
        |cs
        |assign
        |brk
        |comments
        |while0
        |for0
        |check1
        |check2
        |print
        |callfucntion
        ;
```

Switch(check2):

```
check2:SWTCH LPAR expr RPAR NEWLINE cs default ESWTCH NEWLINE ;

cs:     CS LPAR expr RPAR NEWLINE prog_commands
        ;

default: |DFLT NEWLINE prog_commands
         ;
```

Which uses both the case (cs) and default grammar.

Expr:

```
//expresion syntax
expr :  expr ADD  expr
       | expr MIN  expr
       | expr DIV  expr        {if($3==0)yyerror("\ncan't divide with zero.");}
       | expr MUL  expr
       | LPAR expr RPAR       {$$=$2;}
       | con
       |name
       ;
```

Print:

```
//print syntax
print:PRT LPAR QM value QM RPAR SEMI NEWLINE
        |PRT LPAR QM value QM LAGY COMA name RAGY RPAR SEMI NEWLINE
        ;
```

Value:

```
value:  con      //returned value can be a constant, a string or a name
        |name
        ;
```

Application of 1st input:

```
$ ./parser input2.txt
Enter the filename to open
input2.txt
PROGRAM code
STARTMAIN
VARS
INTEGER num,var;
SWITCH(var+1)
CASE(1)
PRINT("caseone"[,num]);
CASE(2)
PRINT("casetwo"[,num]);
DEFAULT
PRINT("casethree"[,num]);
ENDSWITCH
ENDMAIN
```

2st Input:

```
PROGRAM code
FUNCTION hlikia(age)
VARS
INTEGER age18;
WHILE(age!=0)
IF(age>age18)THEN
PRINT("adult");
ELSE
PRINT("underage");
ENDIF
BREAK;
ENDWHILE
RETURN 0
ENDFUNCTION
```

STARTMAIN

VARS

INTEGER choseage;

choseage=5;

hlikia(choseage);

ENDMAIN

In the above entry, apart from the grammar mentioned above, the grammar of are also used:

Function:

```
//function syntax
function: |DEF name LPAR parameters RPAR NEWLINE declarationstar NEWLINE prog_commands RTRN value NEWLINE EDEF NEWLINE;
```

Parameters:

```
parameters:     |parameters COMA str con
                |parameters COMA str
                |str con
                |str
                ;
```

While:

```
 //while syntax
 while0: WHL LPAR condition RPAR NEWLINE prog_commands EWHL NEWLINE;
```

If (check1):

```
check1:LIF LPAR condition RPAR THN NEWLINE prog_commands EIF NEWLINE
       |LIF LPAR condition RPAR THN NEWLINE prog_commands elseif EIF NEWLINE
       |LIF LPAR condition RPAR THN NEWLINE prog_commands elseif lelse EIF NEWLINE
       |LIF LPAR condition RPAR THN NEWLINE prog_commands lelse EIF NEWLINE
       ;

elseif: LELSELIF LPAR condition RPAR NEWLINE prog_commands;

lelse: LELSE NEWLINE prog_commands;
```

Condition:

```
//condition syntax
condition: condition logexp condition
        | value logexp  value
        |value
        ;
//logical operators
logexp: LLE
        |RLE
        |EQUL
        |NOTEQ
        |AND
        |OR
        ;
```

Break:

```
//break syntax
brk: BRK SEMI NEWLINE
```

Assign:

```
assign: name ASSGN expr SEMI NEWLINE
        |name array ASSGN expr SEMI NEWLINE//assign syntax
        ;
```

Callfunction:

```
callfucntion:name LPAR parameters RPAR SEMI NEWLINE;
```

The syntax callfunction is not listed in the statement of the task and is the syntax used to call the defined function.

Application of the 2nd input:

```
$ ./parser input3.txt
Enter the filename to open
input3.txt
PROGRAM code
FUNCTION hlikia(age)
VARS
INTEGER age18;
WHILE(age!=0)
IF(age>age18)THEN
PRINT("adult");
ELSE
PRINT("underage");
ENDIF
BREAK;
ENDWHILE
RETURN 0
ENDFUNCTION
STARTMAIN
VARS
INTEGER choseage;
choseage=5;
hlikia(choseage);
ENDMAIN
```

2) User data type declaration / Structure (struct) declaration:

It starts with the reserved word STRUCT followed by the type name of the struct. We have line change and variable definition as described above. The grammar closes with the reserved word ENDSTRUCT. Alternatively, this grammar can be used with the word TYPEDEF before the word STRUCT and the type name before the word ENDSTRUCT. In the flex word parser we added the bound words "STRUCT, ENDSTRUCT, TYPEDEF", which it returns as STR, ENDSTR, TPDEF respectively. In the syntactic parser bison we bound them with the tokens STR, ENDSTR, TPDEF.

```
//struct syntax
struct:     |struct STR name NEWLINE declarationstar NEWLINE ENDSTR NEWLINE
            |struct TPDEF STR name NEWLINE declarationstar name NEWLINE ENDSTR NEWLINE
            | STR name NEWLINE declarationstar NEWLINE ENDSTR NEWLINE
            | TPDEF STR name NEWLINE declarationstar name NEWLINE ENDSTR NEWLINE
            ;
```

3) Check that the variables used anywhere in the program are correctly declared: Not implemented.

4) Multi-line comments:

```
%x C_COMMENT

%%

"/*"            { BEGIN(C_COMMENT); }
<C_COMMENT>"*/" { BEGIN(INITIAL); }
<C_COMMENT>\n   { }
<C_COMMENT>.    { }
```

The third input uses the grammar of questions 2 and 4 in combination with the grammar implemented in question 1.

3rd Input:

PROGRAM code

STRUCT str1

VARS

INTEGER array1[2] /* comment */ INTEGER c1,c2 CHAR c3,c4;

ENDSTRUCT

FUNCTION func(a,b,par)

VARS

INTEGER v1,v2 CHAR bathmos,v4 INTEGER array[1];

```
IF(y>5)THEN
bathmos=1;
ELSE
batmos=0;
ENDIF
RETURN 0
ENDFUNCTION
STARTMAIN
VARS
INTEGER v1,v2 CHAR v3,v4 INTEGER array[1];
FOR COUNTER = 1 TO 5 STEP 1
PRINT("BATHMOLOGIES"[,array]);
ENDFOR
WHILE(v1!=0)
func(v1,v2,v3);
v4=(v1+v2+v3)/2;
ENDWHILE
ENDMAIN
```

In the above input you also use the for command, which is not reused in the previous inputs with syntax:

```
for0:FR CNTR ASSGN con  LTO  con STP expr NEWLINE prog_commands EFR NEWLINE;
```

Application of the 3rd input:

```
$ ./parser input.txt
Enter the filename to open
input.txt
PROGRAM code
STRUCT str1
VARS
INTEGER array1[2] /* comment */ INTEGER c1,c2 CHAR c3,c4;
ENDSTRUCT
FUNCTION func(a,b,par)
VARS
INTEGER v1,v2 CHAR bathmos,v4 INTEGER array[1];
IF(y>5)THEN
bathmos=1;
ELSE
batmos=0;
ENDIF
RETURN 0
ENDFUNCTION
STARTMAIN
VARS
INTEGER v1,v2 CHAR v3,v4 INTEGER array[1];
FOR COUNTER = 1 TO 5 STEP 1
PRINT("BATHMOLOGIES"[,array]);
ENDFOR
WHILE(v1!=0)
func(v1,v2,v3);
v4=(v1+v2+v3)/2;
ENDWHILE
ENDMAIN
```