**ΕΡΓΑΣΙΑ FLEX/BISON – ΣΕΠΤΕΜΒΡΗΣ 2021**

Πουρνάρας Κωνσταντίνος 1054420 5ο [up1054420@upnet.gr](mailto:up1054420@upnet.gr)

Σουβερμεζλής Μιχαήλ 1054434 5ο [up1054434@upnet.gr](mailto:up1054434@upnet.gr)

BNF :

<PROGRAM> ::= <PROGRAMHEAD> <PROGRAMBODY>

<PROGRAMHEAD> ::= "PROGRAM" <ID> "NL"

<PROGRAMBODY> ::= <FUNCTION> <MAIN>

<FUNCTION> ::= <FUNCTIONHEADER> "NL" <FUNCTIONBLOCK> | E

<FUNCTIONHEADER> ::= "FUNCTION" <ID> <PARAMETERS>

<PARAMETERS> ::= <ID> <M>

<FUNCTIONBLOCK> ::= "VARS" <VARDECLERATION> ";" <BLOCK\_CODE> "RETURN" <RETURNN> "END\_FUNCTION"

<VARDECLERATION> ::= <VARS> <M> ";" <VARDECLERATION> | E

<VARS> ::= <DATATYPE> <ID>

<DATATYPE> ::= "INT" | "CHAR"

<M> ::= "," <ID> <M> | E

<RETURNN> ::= <ID> | <VARS>

<MAIN> ::= "STARMAIN" <VARS> <BLOCK\_CODE> "ENDMAIN"

<BLOCK\_CODE> ::= <ANATHESHS> | <BROXOU> | <ELEGXOU> | <EKTUPOSH> | <TERMATISMOU> | <SXOLIA> | E

<ANATHESHS> ::= <VARS> "=" <EKFRASH>

<EKFRASH> ::= <VARS> | <ID> | <NUM> | <EKFRASH> <ARITHMETICEXP> <EKFRASH> | "(" <EKFRASH> ")" | "-" <EKFRASH>

<ARITHMETICEXP> ::= "+" | "-" | "^" | "\*" | "/"

<BROXOU> ::= "WHILE" <CONDITION> <BLOCK\_CODE> "ENDWHILE" | "FOR" <ID> ":=" <NUM> "TO" <NUM> "STEP" <NUM> <BLOCK\_CODE> "ENDFOR"

<CONDITION> ::= <VARS> | <ID> | <CONDITION> <OPERATOR> <CONDITION> | "(" <CONDITION> ")" | "-" <CONDITION>

<OPERATOR> ::= "<" | ">" | "==" | "!=" | "AND" | "OR"

<ELEGXOU> ::= "IF" <CONDITION> "THEN" <BLOCK\_CODE> <ELSEIF> <ELSE> "ENDIF" | "SWITCH" <CONDITION> <CASE> <DEFAULTCASE> "ENDSWITCH"

<ELSEIF> ::= "ELSEIF" <EKFRASH> "THEN" <BLOCK\_CODE> <ELSEIF> | E

<ELSE> ::= "ELSE" <BLOCK\_CODE> | E

<CASE> ::= "CASE" <EKFRASH> ":" <BLOCK\_CODE> <CASE> | "CASE" <EKFRASH> ":" <BLOCK\_CODE>

<DEFAULTCASE> ::= "DEFAULT:" <BLOCK\_CODE> | E

<EKTUPOSH> ::= "PRINT" "(" <TEXT> "[," <VARS> "]" ");"

<TERMATISMOU> ::= "BREAK;"

<SXOLIA> ::= "%" "NL"

<TEXT> ::= " ' " [a-z] [A-Z] [0-9] " " "SYMBOLS" " ' "

<ID> ::= [a-z] [A-Z] [0-9] "\_"

<NUM> ::= [0-9]

Ask.l

%option noyywrap

%option yylineno

%{

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "ask.tab.h"

#include <stdbool.h>

bool need\_nl(void);

void yyerror();

%}

alpha [a-zA-Z]

digit [0-9]

alnum {alpha}|{digit}

ID {alpha}+{alnum}\*

INTEGER (int)

CHAR (char)

%x ML\_COMMENT

%%

"PROGRAM" {return PROGRAMM;}

";" {return SEM;}

"VARS" {return VAR;}

"CHAR" {return CHARR;}

"INTEGER" {return INTT;}

"FUNCTION" {return FUN;}

"END\_FUNCTION" {return ENDFUN;}

"IF" {return ENDIFF;}

"ENDIF" {return IFF;}

"ELSEIF" {return ELSEIFF;}

"ELSE" {return ELSEE;}

"THEN" {return THENN;}

"SWITCH" {return SWITCHH;}

"ENDSWITCH" {return ENDSWITCHH;}

"CASE" {return CASEE;}

"DEFAULT" {return DEFAULL;}

"WHILE" {return WHILEE;}

"ENDWHILE" {return ENDWHILEE;}

"FOR" {return FORR;}

"ENDFOR" {return ENDFORR;}

"TO" {return TTO;}

"STEP" {return STEPP;}

"STARTMAIN" {return STARTTMAIN;}

"ENDMAIN" {return ENDDMAIN;}

"RETURN" {return RETURNNN;}

"PRINT" {return PRINTT;}

"BREAK" {return BREAKK;}

"STRUCT" {return STRUCTT;}

"ENDSTRUCT" {return ENDSTRUCTT;}

"TYPEDEF" {return TYPEDEFF;}

":=" {return FORREQUALL;}

"%" {return COMMENTT;}

"=" {return ASS;}

":" {return PANKAT;}

"," {return COMMA;}

"(" {return LPAR;}

")" {return RPAR;}

"[" {return LBR;}

"]" {return RBR;}

"AND" {return ANDD;}

"OR" {return ORR;}

"+" {return PLUS;}

"-" {return MINUS;}

"^" {return POWERR;}

"\*" {return MULT;}

"/" {return DIVV;}

"==" {return EQUAL;}

"!=" {return DIFF;}

"<" {return SM;}

">" {return BIG;}

[0-9]+ {

yylval.num = atoi(yytext);

return NUM;

}

([[:space:]]{-}[\n])+ { }

{ID} {

yylval.id = strdup(yytext);

return ID;

}

<INITIAL>"/\*" BEGIN(ML\_COMMENT);

<ML\_COMMENT>"\*/" BEGIN(INITIAL);

<ML\_COMMENT>. | "\n" { }

[[:space:]]+ { if (need\_nl()) return '\n';}

. { yyerror("Unrecognized character"); }

%%

void yyerror()

{

fprintf(stderr, "Syntax error at line %d\n", yylineno);

exit(1);

}

Ask.y

%{

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#include <stdbool.h>

void yyerror();

extern char \*yytext;

extern int yylex();

extern FILE \*yyin;

bool need\_nl(void);

static bool need\_nl\_flag = false;

void Codeprnt(char \* arg);

%}

%union

{

int num;

char \*id;

}

%token PROGRAMM

%token SEM

%token VAR

%token <num> INTT

%token <id> CHARR

%token FUN

%token ENDFUN

%token IFF

%token ENDIFF

%token ELSEIFF

%token THENN

%token ELSEE

%token SWITCHH

%token ENDSWITCHH

%token CASEE

%token DEFAULL

%token WHILEE

%token ENDWHILEE

%token FORR

%token ENDFORR

%token TTO

%token STEPP

%token STARTTMAIN

%token ENDDMAIN

%token PRINTT

%token RETURNNN

%token BREAKK

%token <id> STRUCTT

%token <id> ENDSTRUCTT

%token <id> TYPEDEFF

%token FORREQUALL

%token COMMENTT

%token ASS

%token PANKAT

%token COMMA

%token LPAR

%token RPAR

%token LBR

%token RBR

%token ANDD

%token ORR

%token PLUS

%token MINUS

%token POWERR

%token MULT

%token DIVV

%token EQUAL

%token DIFF

%token SM

%token BIG

%token ID

%token <int> NUM

%type <id> ID

%type EKFRASH

%type VALUE

%type <id> STRUCTDEFTWO

%start PROGRAM

%%

/\* Program directive. \*/

PROGDECL : PROGRAMM { need\_nl\_flag = true; } ID '\n';

/\* Function definition \*/

FUNDEF : FUN ID LPAR PARAMETERS { need\_nl\_flag = true; } RPAR'\n';

/\*grammar rules\*/

PROGRAM : PROGDECL PROGRAMBODY ;

PROGRAMBODY : STRUCT FUNCTION MAIN;

STRUCT : STRUCTDEF STRUCTBODY | STRUCTDEFTWO | ;

STRUCTDEF : STRUCTT { need\_nl\_flag = true; } ID'\n';

STRUCTBODY : VARDECLERATION ENDSTRUCTT;

STRUCTDEFTWO : TYPEDEFF STRUCTT { need\_nl\_flag = true; } ID '\n' ID ENDSTRUCTT

{

if(strcmp($4, $6) != 0 )

{

printf("Struct definition different names %s and %s \n", $4, $6);

exit(0);

}

};

FUNCTION : FUNDEF FUNCTIONBLOCK | ;

PARAMETERS : VARS A | ;

A : COMMA VARS A | ;

FUNCTIONBLOCK : VARDECLERATION BLOCKCODE RETURNNN RETURNN ENDFUN ;

MAIN : MAINHEADER MAINBODY ;

MAINHEADER : STARTTMAIN ;

MAINBODY : VARDECLERATION BLOCKCODE ENDDMAIN ;

VARDECLERATION : VAR VARDEC | ;

VARDEC : VARS ARRAY M VARDEC | ;

ARRAY : LBR NUM RBR | ;

VARS : DATATYPE ID ;

DATATYPE : INTT | CHARR ;

M : COMMA ID ARRAY M | SEM ;

RETURNN : ID | VARS | NUM ;

BLOCKCODE : CODE BLOCKCODE | ;

CODE : ANATHESHS | BROXOU | ELEGXOU | EKTUPOSH | TERMATISMOU | SXOLIA ;

ANATHESHS : ID ASS EKFRASH SEM ;

EKFRASH : VALUE ARITHMETICEXP EKFRASH | VALUE ;

ARITHMETICEXP : PLUS | MINUS | POWERR | MULT | DIVV ;

BROXOU : WHILEE CONDITION BLOCKCODE ENDWHILEE | FORR ID FORREQUALL NUM TTO NUM STEPP NUM BLOCKCODE ENDFORR ;

CONDITION : VALUE OPERATOR CONDITION | VALUE ;

OPERATOR : SM | BIG | EQUAL | DIFF | ANDD | ORR ;

ELEGXOU : IF | SWITCH ;

IF : IFF CONDITION THENN BLOCKCODE ELSEIF ELSE ENDIFF ;

SWITCH : SWITCHH CONDITION CASE DEFAULTCASE ENDSWITCHH ;

ELSEIF : ELSEIFF EKFRASH THENN BLOCKCODE ELSEIF | ;

ELSE : ELSEE BLOCKCODE | ;

CASE : CASEE EKFRASH PANKAT BLOCKCODE CASE | CASEE EKFRASH PANKAT BLOCKCODE ;

DEFAULTCASE : DEFAULL PANKAT BLOCKCODE | ;

EKTUPOSH : PRINTT LPAR VALUE LBR COMMA VARS RBR RPAR SEM ;

TERMATISMOU : BREAKK SEM ;

SXOLIA : COMMENTT { need\_nl\_flag = true; } ID '\n';

VALUE : ID | NUM ;

%%

bool need\_nl(void) {

bool temp = need\_nl\_flag;

need\_nl\_flag = false;

return temp;

}

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

//#ifdef YYDEBUG

//yydebug = 1;

//#endif

int flag;

//elegxoume an exoume 2o orisma, dld an dothike onoma arxeiou c

if(argc==2)

{

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

}

else

{

yyin = stdin;

}

flag = yyparse();

Codeprnt(argv[1]);

printf("No syntax error. \n");

return flag;

}

void Codeprnt(char \* arg)

{

char c;

FILE \*fptr;

fptr = fopen(arg, "r");

if (fptr == NULL)

{

printf("Cannot open file \n");

exit(0);

}

c = fgetc(fptr);

while (c != EOF)

{

printf ("%c", c);

c = fgetc(fptr);

}

fclose(fptr);

}





