YACC Documentation

```
%{
   #include <stdio.h>
   #include <stdlib.h>
   #include <string.h>
   #define YYDEBUG 1
   #define MAX 100 /*to store productions*/
   char productions[MAX][MAX];
   int count = 1;
%}

%token ID 1
%token CONST 2
%token osszead 3
%token kivon 4
%token szoroz 5
%token oszt 6
%token modulo 7
%token kisebb 8
%token kisebbvagyegyenlo 9
%token egyenlo 10
%token nagyobbvagyegyenlo 11
%token nagyobb 12
%token nemegyenlo 13
%token novel 14
%token csokken 15
%token kapja 16
%token ha 17
%token kulonben 18
```

%token karakter 19

%token karakterlanc 20

%token amig 21

%token boolean 22

%token egesz 23

%token tomb 24

%token dupla 25

%token visszater 26

%token ismeteld 27

%token Kezd 28

%token Vegez 29

%token allj 30

%token valassz 31

%token eset 32

%token alapertelmezett 33

%token konstans 34

%token open_square_bracket 35

%token closed_square_bracket 36

%token open_curly_bracket 37

%token closed_curly_bracket 38

%token open_bracket 39

%token closed_bracket 40

%token semicolon 41

%token coma 42

%token colon 43

%token olvas 44

%token kiir 45

%start program

%%

program : compoundstatement {strcpy(productions[count++],"program");} ;

compoundstatement : Kezd statementlist Vegez {strcpy(productions[count++],"compoundstatement");}
;

statementlist : statement semicolon statementlist {strcpy(productions[count++],"statement semicolon
statementlist");} | statement {strcpy(productions[count++],"statement");} ;

statement : declarationlist {strcpy(productions[count++],"declarationlist");} | simplestatement
{strcpy(productions[count++],"simplestatement");} | structuredtatement
{strcpy(productions[count++],"structuredtatement");} ;

declarationlist : type identifierList {strcpy(productions[count++],"type identifierList");} ;

identifierList : ID coma identifierList {strcpy(productions[count++],"ID coma identifierList");} | ID
{strcpy(productions[count++],"ID");} ;

type : egesz {strcpy(productions[count++],"egesz");} | dupla {strcpy(productions[count++],"dupla");} |
karakter {strcpy(productions[count++],"karakter");} | karakterlanc
{strcpy(productions[count++],"karakterlanc");} | boolean {strcpy(productions[count++],"boolean");} ;

simplestatement : assignment {strcpy(productions[count++],"assignment");} | iostatement
{strcpy(productions[count++],"iostatement");} ;

assignment : ID kapja expression {strcpy(productions[count++],"ID kapja expression");} ;

expression : term {strcpy(productions[count++],"term");} | term additive expression
{strcpy(productions[count++],"term additive expression");} ;

term : factor {strcpy(productions[count++],"factor");} | term multiplicative factor
{strcpy(productions[count++],"term multiplicative factor");} ;

factor : open_bracket expression closed_bracket {strcpy(productions[count++],"open_bracket
expression closed_bracket");} | ID {strcpy(productions[count++],"ID");} | CONST
{strcpy(productions[count++],"CONST");} ;

additive : osszead {strcpy(productions[count++],"osszead");} | kivon
{strcpy(productions[count++],"kivon");} ;

multiplicative : oszt {strcpy(productions[count++],"oszt");} | szoroz
{strcpy(productions[count++],"szoroz");} | modulo {strcpy(productions[count++],"modulo");} ;

iostatement : olvas open_bracket value closed_bracket {strcpy(productions[count++],"olvas
open_bracket value closed_bracket");} | kiir open_bracket value closed_bracket
{strcpy(productions[count++],"kiir open_bracket value closed_bracket");} ;

value : ID {strcpy(productions[count++],"ID");} | CONST {strcpy(productions[count++],"CONST");} ;

structuredtatement : ifstatement {strcpy(productions[count++],"ifstatement");} | whilestatement {strcpy(productions[count++],"whilestatement");} | forstatement {strcpy(productions[count++],"forstatement");} ;

ifstatement : ha open_bracket condition closed_bracket open_curly_bracket statement closed_curly_bracket {strcpy(productions[count++],"ha open_bracket condition closed_bracket open_curly_bracket statement closed_curly_bracket");} | ha open_bracket condition closed_bracket open_curly_bracket statement closed_curly_bracket kulonben open_curly_bracket statement closed_curly_bracket {strcpy(productions[count++],"ha open_bracket condition closed_bracket open_curly_bracket statement closed_curly_bracket kulonben open_curly_bracket statement closed_curly_bracket");} ;

condition : expression relation expression {strcpy(productions[count++],"expression relation expression");} ;

whilestatement : amig open_bracket condition closed_bracket open_curly_bracket statementlist closed_curly_bracket {strcpy(productions[count++],"amig open_bracket condition closed_bracket open_curly_bracket statementlist closed_curly_bracket");} ;

forstatement : ismeteld open_bracket statement semicolon statement semicolon statement closed_bracket open_curly_bracket statement closed_curly_bracket {strcpy(productions[count++],"ismeteld open_bracket statement semicolon statement semicolon statement closed_bracket open_curly_bracket statement closed_curly_bracket");} ;

relation : nemegyenlo {strcpy(productions[count++],"nemegyenlo");} | egyenlo {strcpy(productions[count++],"egyenlo");} | nagyobbvagyegyenlo {strcpy(productions[count++],"nagyobbvagyegyenlo");} | kisebbvagyegyenlo {strcpy(productions[count++],"kisebbvagyegyenlo");} | kisebb {strcpy(productions[count++],"kisebb");} | nagyobb {strcpy(productions[count++],"nagyobb");} ;

%%

```
int yyerror(char *s)
{
  printf("%s\n", s);
}

extern FILE *yyin;

int main(int argc, char **argv)
{
  if (argc > 1)
    yyin = fopen(argv[1], "r");
  if ((argc>2) && (!strcmp(argv[2],"-d")))
    yydebug = 1;
  if (!yyparse()) {
    fprintf(stderr,"Successful parsing !!!\n");
    printf("String of productions:\n");
    printf("============================================================\n");
    for(int i = count-1; i > 0; i--){
      printf("%s\n",productions[i]);
    }
    printf("============================================================\n");
  }
}
```

LEX Documentation

```
%{
#include <math.h>
#include <stdio.h>
#include "y.tab.h"
int current_line = 0;

%}

%option noyywrap

DIGIT               [0-9]
STRING              \".*\"
INTEGER                     [-+]?[1-9][0-9]*|0
REAL_NUMBER             {INTEGER}+"."{DIGIT}*
CONSTANT_VALUE                  {STRING}|{INTEGER}|{REAL_NUMBER}
IDENTIFIER                  [a-zA-Z][a-zA-Z0-9_]{0,256}

%%

"osszead"           {printf( "Reserved word: %s\n", yytext );return osszead;}
"kivon" {printf( "Reserved word: %s\n", yytext );return kivon;}
"szoroz"            {printf( "Reserved word: %s\n", yytext );return szoroz;}
"oszt"    {printf( "Reserved word: %s\n", yytext );return oszt;}
"modulo"            {printf( "Reserved word: %s\n", yytext );return modulo;}
"kisebb"            {printf( "Reserved word: %s\n", yytext );return kisebb;}
"kisebbvagyegyenlo"     {printf( "Reserved word: %s\n", yytext );return kisebbvagyegyenlo;}
"egyenlo"           {printf( "Reserved word: %s\n", yytext );return egyenlo;}
"nagyobbvagyegyenlo"  {printf( "Reserved word: %s\n", yytext );return nagyobbvagyegyenlo;}
```

```
"nagyobb"        {printf( "Reserved word: %s\n", yytext );return nagyobb;}
"nemegyenlo"   {printf( "Reserved word: %s\n", yytext );return nemegyenlo;}
"novel" {printf( "Reserved word: %s\n", yytext );return novel;}
"csokken"        {printf( "Reserved word: %s\n", yytext );return csokken;}
"kapja" {printf( "Reserved word: %s\n", yytext );return kapja;}
"ha"     {printf( "Reserved word: %s\n", yytext );return ha;}
"kulonben"       {printf( "Reserved word: %s\n", yytext );return kulonben;}
"karakter"       {printf( "Reserved word: %s\n", yytext );return karakter;}
"karakterlanc"   {printf( "Reserved word: %s\n", yytext );return karakterlanc;}
"amig"  {printf( "Reserved word: %s\n", yytext );return amig;}
"boolean"        {printf( "Reserved word: %s\n", yytext );return boolean;}
"egesz" {printf( "Reserved word: %s\n", yytext );return egesz;}
"tomb" {printf( "Reserved word: %s\n", yytext );return tomb;}
"dupla" {printf( "Reserved word: %s\n", yytext );return dupla;}
"visszater"      {printf( "Reserved word: %s\n", yytext );return visszater;}
"ismeteld"       {printf( "Reserved word: %s\n", yytext );return ismeteld;}
"Kezd"  {printf( "Reserved word: %s\n", yytext );return Kezd;}
"Vegez"{printf( "Reserved word: %s\n", yytext );return Vegez;}
"allj"     {printf( "Reserved word: %s\n", yytext );return allj;}
"valassz"        {printf( "Reserved word: %s\n", yytext );return valassz;}
"eset"   {printf( "Reserved word: %s\n", yytext );return eset;}
"alapertelmezett"        {printf( "Reserved word: %s\n", yytext ); return alapertelmezett;}
"konstans"       {printf( "Reserved word: %s\n", yytext ); return konstans;}
"olvas"  {printf( "Reserved word: %s\n", yytext ); return olvas;}
"kiir"     {printf( "Reserved word: %s\n", yytext ); return kiir;}
{IDENTIFIER}     {printf( "Identifier: %s\n", yytext ); return ID;}
{CONSTANT_VALUE}     {printf( "Constant: %s\n", yytext ); return CONST;}
"["      {printf( "Separator: %s\n", yytext );return open_square_bracket;}
"]"      {printf( "Separator: %s\n", yytext );return closed_square_bracket;}
```

```
"{"         {printf( "Separator: %s\n", yytext );return open_curly_bracket;}

"}"         {printf( "Separator: %s\n", yytext );return closed_curly_bracket;}

"("         {printf( "Separator: %s\n", yytext );return open_bracket;}

")"         {printf( "Separator: %s\n", yytext );return closed_bracket;}

";"         {printf( "Separator: %s\n", yytext );return semicolon;}

","         {printf( "Separator: %s\n", yytext );return coma;}

":"         {printf( "Separator: %s\n", yytext );return colon;}

"{"[^}\n]*"}"      /* eliminate the comments in the code*/ {}

[ \t]+     /* eliminate the spaces in the code */     {}

[\n]+     {++current_line;}

[a-zA-Z][a-zA-Z0-9]{256,}            {printf("Illegal size of the identifier %s at line %d\n",yytext,
current_line); return -1;}

[0-9][a-zA-Z0-9]{0,256}  {printf("Illegal identifier %s at line %d\n",yytext, current_line); return -1;}

.           {printf("Illegal symbol %s at line %d\n",yytext,current_line); return -1;}

%%
```