https://github.com/Arnold-21/Compiler/tree/main

```
%{
#include <stdio.h>
#include <stdlib.h>

#define YYDEBUG 1
%}

%token ARR;
%token INT;
%token BOOL;
%token CHAR;
%token STRING;
%token IF;
%token ELSE;
%token WHILE;
%token PRINT;
%token READINT;
%token READSTRING;
%token SET;
%token GET;

%token BOOLCONST;
%token CHARCONST;
%token STRINGCONST;
%token IDENTIFIER;
%token INTCONST;

%token PLUS;
%token MINUS;
%token TIMES;
%token DIV;
%token MOD;
%token EQ;
%token BIGGER;
%token BIGGEREQ;
%token LESS;
%token LESSEQ;
%token EQQ;
%token NEG;
%token AND;
%token OR;

%token SEMICOLON;
%token OPEN;
%token CLOSE;
%token SOPEN;
%token SCLOSE;
%token BRACKETOPEN;
%token BRACKETCLOSE;
%token COMMA;
%token QUOTE;
%token SIMPLEQUOTE;
```

```
%start program

%%

program : declaration statement {printf("program -> declaration statement\n");} ;
declaration : simpledeclaration SEMICOLON declaration {printf("declaration -> simpledeclaration ; declara
tion\n");} ;
        | arraydeclaration SEMICOLON declaration {printf("declaration -> arraydeclaration ; declaration\n")
;};
        |   {printf("declaration -> epsilon\n");} ;
simpledeclaration : type identifierlist {printf("simpledeclaration -> type identifierlist\n");} ;
type : INT {printf("type -> int\n");} ;
    | BOOL {printf("type -> bool\n");} ;
    | CHAR {printf("type -> char\n");} ;
    | STRING {printf("type -> string\n");} ;
identifierlist : IDENTIFIER {printf("identifierlist -> IDENTIFIER\n");} ;
            | IDENTIFIER EQ expression {printf("identifierlist -> IDENTIFIER = expression\n");} ;
            | IDENTIFIER COMMA identifierlist {printf("identifierlist -> IDENTIFIER , identifierlist\n");} ;
            | IDENTIFIER EQ expression COMMA identifierlist {printf("identifierlist -> IDENTIFIER = expres
sion\n");} ;
expression : intexpression {printf("expression -> intexpression\n");} ;
        | boolexpression {printf("expression -> boolexpression\n");} ;
        | charexpression {printf("expression -> charexpression\n");} ;
        | stringexpression {printf("expression -> stringexpression\n");} ;
simpleintexpression : INTCONST {printf("simpleintexpression -> INTCONST\n");} ;
                | IDENTIFIER {printf("simpleintexpression -> IDENTIFIER\n");} ;
intexpression : simpleintexpression {printf("intexpression -> simpleintexpression\n");} ;
            | OPEN simpleintexpression TIMES intexpression CLOSE {printf("intexpression -> ( simpleintex
pression * intexpression )\n");} ;
            | OPEN simpleintexpression DIV intexpression CLOSE {printf("intexpression -> ( simpleintexpre
ssion / intexpression )\n");} ;
            | OPEN simpleintexpression MOD intexpression CLOSE {printf("intexpression -> ( simpleintexpr
ession mod intexpression )\n");} ;
            | OPEN simpleintexpression PLUS intexpression CLOSE {printf("intexpression -> ( simpleintexp
ression + intexpression )\n");} ;
            | OPEN simpleintexpression MINUS intexpression CLOSE {printf("intexpression -> ( simpleintex
pression - intexpression )\n");} ;
            | simpleintexpression TIMES intexpression {printf("intexpression -> simpleintexpression * intexp
ression\n");} ;
            | simpleintexpression DIV intexpression {printf("intexpression -> simpleintexpression / intexpres
sion\n");} ;
            | simpleintexpression MOD intexpression {printf("intexpression -> simpleintexpression mod intex
pression\n");} ;
            | simpleintexpression PLUS intexpression {printf("intexpression -> simpleintexpression + intexpr
ession\n");} ;
            | simpleintexpression MINUS intexpression {printf("intexpression -> simpleintexpression - intexp
ression\n");} ;
simpleboolexpression : BOOLCONST {printf("simpleboolexpression -> BOOLCONST\n");} ;
                    | NEG IDENTIFIER {printf("simpleboolexpression -> ! IDENTIFIER\n");} ;
                    | IDENTIFIER {printf("simpleboolexpression -> IDENTIFIER\n");} ;
boolexpression : simpleboolexpression {printf("boolexpression -> simpleboolexpression\n");} ;
            | OPEN simpleboolexpression AND boolexpression CLOSE {printf("boolexpression -> ( simpleb
oolexpression && boolexpression )\n");} ;
            | OPEN simpleboolexpression OR boolexpression CLOSE {printf("boolexpression -> ( simplebo
olexpression || boolexpression )\n");} ;
```

```
            | OPEN simpleboolexpression EQQ boolexpression CLOSE {printf("boolexpression -> ( simpleb
oolexpression == boolexpression )\n");} ;
            | OPEN intexpression EQQ intexpression CLOSE {printf("boolexpression -> ( intexpression == i
ntexpression )\n");} ;
            | OPEN intexpression LESS intexpression CLOSE {printf("boolexpression -> ( intexpression < in
texpression )\n");} ;
            | OPEN intexpression LESSEQ intexpression CLOSE {printf("boolexpression -> ( intexpression
<= intexpression )\n");} ;
            | OPEN intexpression BIGGER intexpression CLOSE {printf("boolexpression -> ( intexpression
> intexpression )\n");} ;
            | OPEN intexpression BIGGEREQ intexpression CLOSE {printf("boolexpression -> ( intexpressi
on >= intexpression )\n");} ;
            | simpleboolexpression AND boolexpression {printf("boolexpression -> simpleboolexpression &
& boolexpression\n");} ;
            | simpleboolexpression OR boolexpression {printf("boolexpression -> simpleboolexpression || b
oolexpression\n");} ;
            | simpleboolexpression EQQ boolexpression {printf("boolexpression -> simpleboolexpression =
= boolexpression\n");} ;
            | intexpression EQQ intexpression {printf("boolexpression -> intexpression == intexpression\n");
} ;
            | intexpression LESS intexpression {printf("boolexpression -> intexpression < intexpression\n");}
 ;
            | intexpression LESSEQ intexpression {printf("boolexpression -> intexpression <= intexpression\
n");} ;
            | intexpression BIGGER intexpression {printf("boolexpression -> intexpression > intexpression\n
");} ;
            | intexpression BIGGEREQ intexpression {printf("boolexpression -> intexpression >= intexpressi
on\n");} ;
charexpression : CHARCONST {printf("charexpression -> CHARCONST\n");} ;
simplestringexpression : STRINGCONST {printf("simplestringexpression -> STRINGCONST\n");} ;
                       | IDENTIFIER {printf("simplestringexpression -> IDENTIFIER\n");} ;
stringexpression : simplestringexpression {printf("stringexpression -> simplestringexpression\n");} ;
             | OPEN simplestringexpression PLUS stringexpression CLOSE {printf("stringexpression -> ( si
mplestringexpression + stringexpression )\n");} ;
             | simplestringexpression PLUS stringexpression {printf("stringexpression -> simplestringexpres
sion + stringexpression\n");} ;
arraydeclaration : ARR SOPEN type SCLOSE SOPEN INTCONST SCLOSE simpleidentifierlist {printf("arr
aydeclaration -> array [ type ] [ INTCONST ] simpleidentifierlist\n");} ;
simpleidentifierlist : IDENTIFIER {printf("simpleidentifierlist -> IDENTIFIER\n");} ;
                     | IDENTIFIER COMMA simpleidentifierlist {printf("simpleidentifierlist -> IDENTIFIER , simplei
dentifierlist\n");} ;
statement : assignstatement SEMICOLON statement {printf("statement -> assignstatement ; statement\n"
);} ;
          | ifstatement statement {printf("statement -> ifstatement statement\n");} ;
          | whilestatement statement {printf("statement -> whilestatement statement\n");} ;
          | functionstatement SEMICOLON statement {printf("statement -> functionstatement ; statement\n")
;} ;
          |   {printf("statement -> epsilon\n");} ;
assignstatement : IDENTIFIER EQ expression {printf("assignstatement -> IDENTIFIER = expression\n");}
;
             | IDENTIFIER EQ functionstatement {printf("assignstatement -> IDENTIFIER = functionstateme
nt\n");} ;
ifstatement : IF OPEN boolexpression CLOSE BRACKETOPEN statement BRACKETCLOSE {printf("ifsta
tement -> if ( boolexpression ) { statement }\n");} ;
          | IF OPEN boolexpression CLOSE BRACKETOPEN statement BRACKETCLOSE ELSE BRACK
```

```
ETOPEN statement BRACKETCLOSE {printf("ifstatement -> if ( boolexpression ) { statement } else { stat
ement }\n");} ;
whilestatement : WHILE OPEN boolexpression CLOSE BRACKETOPEN statement BRACKETCLOSE {p
rintf("whilestatement -> while ( boolexpression ) { statement }\n");} ;
functionstatement : functionname OPEN expressionlist CLOSE {printf("functionstatement -> functionname
 ( expressionlist )\n");} ;
                  | functionname OPEN CLOSE {printf("functionstatement -> functionname (  )\n");} ;
functionname : READINT {printf("functionname -> readInt\n");} ;
          | READSTRING {printf("functionname -> readString\n");} ;
          | GET {printf("functionname -> get\n");} ;
          | SET {printf("functionname -> set\n");} ;
          | PRINT {printf("functionname -> print\n");} ;
expressionlist : expression {printf("expressionlist -> expression\n");} ;
               | expression COMMA expressionlist {printf("expressionlist -> expression , expressionlist\n");} ;

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

extern FILE *yyin;

main(int argc, char **argv)
{
 if(argc>1) yyin =  fopen(argv[1],"r");
 if(!yyparse()) fprintf(stderr, "\tOK\n");
}
```