final.l

```
%{
    #include<stdio.h>
    #include<string.h>
    #include <stdlib.h>
    #include <stdbool.h>
    #include <ctype.h>
    #include "final.tab.h"

    // header, single, multi, var, print, dis, calc_val, if, till else
    int tempCounter[11];

    int blockBalancer = 0;
    int stack[100];
    int stackSize = 0;

    const int MAXIMUM_VAR_SIZE = 10;

    char error[200];

    // extern
    char lastDataType[10];
    char *outBuffer = NULL;
    int outBufferSize = 0;
    bool isLastIfValid = false;
    //extern

    int commentCounter = 0;
    int comment_depth = 0;
    char *comment_buffer = NULL;

    bool canDeclareHeader = true;
    bool isMainBlockFound = false;
    bool isCommingFromMain = false;
    bool isCommingFromVar = false;

    size_t buffer_length = 0;

    int bracketCounter=0;

    char *removeRedundant(char*);
    bool getCurrentValidity();
    void insertImport(char*);

    // variable
    const int MAXIMUM_VARIABLE_LENGTH = 200;

    int pushState(int id){
        stackSize++;
        stack[stackSize] = id;
        return id;
    }

    int popState(){
        if(stackSize > 0) stackSize--;
        return stack[stackSize];
    }

    void inc(int index){
        tempCounter[index]++;
    }

    void initSingleComment(){
        commentCounter++;
        printf("\nSingle line comment: %s\n",yytext);
    }

    void appendToBuffer(char ch){
        comment_buffer = (char*) realloc(comment_buffer, buffer_length + 1);
        buffer_length += sprintf(comment_buffer + buffer_length, "%c", ch);
    }

    void appendTextToBuffer(char* ch){

        if( strncmp(ch,"/*",2) == 0) comment_depth++;
        if( strncmp(ch,"*/",2) == 0) comment_depth--;
        comment_buffer = (char*) realloc(comment_buffer, buffer_length + strlen(ch) );
        buffer_length += sprintf(comment_buffer + buffer_length, ch);
    }

    void initMultiComment(){
        comment_depth = 1;
        buffer_length = 2;
        comment_buffer = (char*) malloc(3);

        memset(comment_buffer, 0, 3);
        strcat(comment_buffer, "/*");
    }

    void resetBuffer(){
        free(comment_buffer);
        comment_buffer = NULL;
        buffer_length = 0;
    }

    void process_comment() {
        commentCounter++;
        printf("\nCaptured comment: %s\n\n", comment_buffer);
        //printf("Multi-line comment found\n");
    }

    void stopProgram(char *error){
        printf("%s\n",error);
        free(comment_buffer);
        exit(1);
    }

    void checkForEnd(){
        if (comment_depth == 0) {
            BEGIN(INITIAL);
            process_comment();
            resetBuffer();
        }
        else if(comment_depth < 0){
            resetBuffer();
            stopProgram("Invalid comment found");
        }
    }

    void initMain(){
        canDeclareHeader = false;
        //printf("execution started\n");
    }

    void processHeader(){
        if(!canDeclareHeader){
            stopProgram("Header must be at top");
            return;
        }

    }

    void initVarSec(char *temp){
        char *type = removeRedundant(temp);
        strcpy(lastDataType,type);
        canDeclareHeader = false;
    }

    void sendNumber(char *text){
        double num = 0;
        sscanf(text, "%lf", &num);
        yylval.num = num;
        //printf("======%lf=====",num);
    }

    void initOutBuffer(){
        outBuffer = (char*) malloc(1);
        memset(outBuffer, 0, 1);
        outBufferSize = 1;
    }

    char* removeLpRp(char *val){
        char *op = malloc(14*sizeof(char));
        char *temp = strdup(val);
        strncpy(op,temp+1, strlen(temp) - 2 );
        return op;
    }

%}

%x COMMENT
%x MAIN
%x VAR_SEC
%x IF_SEC
%x ELSE_SEC
%x IGNORE_SEC
%x DISCARD_SEC
%x LOOP_SEC
%x LOOP_BODY_SEC
%x OUT_SEC
%x PROTO_SEC
%x FUNC_SEC
%x FUNC_SEC_VAR

DQ \"
NUMBER [-]?([0-9]+(".")[0-9]+)|[-]?([0-9]+)|("."[0-9]+)
VARIABLE [a-zA-Z][a-zA-Z0-9]*
VAR_NUM ({NUMBER}|{VARIABLE})
COND_OP ("<lt>"|"<gt>"|"<eq>"|"<neq>"|"<le>"|"<ge>")
ARITH_OP ("<add>"|"<sub>"|"<mul>"|"<div>"|"<dif>"|"<rem>")

HEADER "import "[a-zA-Z]+";"

OUT_START "println("[ ]*
OUT_BODY_CONST {DQ}[^\"]*{DQ}
```

```
OUT_BODY_VAR {VAR_NUM}
OUT_SEP [ ]*(","|" ")[ ]*
OUT_END [ ]*")"[ ]*;[ ]*

MAIN_START "static void entryPoint"[ ]*"("[ ]*")"[ ]*"{"[ ]*
MAIN_END "}"

SINGLE_LINE_COMMENT ("//").*(\n)?

VARIABLE_ONLY [ ]*{VARIABLE}*[ ]*(,)
VARIABLE_VALUE [ ]*{VARIABLE}[ ]*("=")[ ]*{NUMBER}(,)
VARIABLE_VALUE_ASSIGN_CONST [ ]*{VARIABLE}[ ]*("=")[ ]*{NUMBER};
VARIABLE_VALUE_ASSIGN_VAR [ ]*{VARIABLE}[ ]*("=")[ ]*{VARIABLE};
VARIABLE_VALUE_ASSIGN_CALC [ ]*{VARIABLE}[ ]*("=")[ ]*{VAR_NUM}[ ]*
{ARITH_OP}[ ]*{VAR_NUM}[ ]*,
VARIABLE_VALUE_ASSIGN_CALC_LAST [ ]*{VARIABLE}[ ]*("=")[ ]*{VAR_NUM}[ ]*
{ARITH_OP}[ ]*{VAR_NUM}[ ]*;

DISCARD_START "discard "

VARIABLE_ONLY_LAST [ ]*[a-zA-Z][a-zA-Z0-9]*[ ]*(;)
VARIABLE_VALUE_LAST [ ]*[a-zA-Z][a-zA-Z0-9]*[ ]*("=")[ ]*{NUMBER}(;)

VAR_SPACE [ ]*

DATA_TYPE ("int"|"float"|"double")[ ]
FUNC_TYPE ("int"|"float"|"double"|"void")

IF "justInCase"[ ]*
ELSE "otherwise"
IF_BODY_START "{"
IF_BODY_END [ ]*"}"
IF_SPACE [ ]*

IGNORE_LEFT_BRACE "{"
IGNORE_RIGHT_BRACE "}"

FUNC_START "@"{VARIABLE}

LOOP_START "till"
LOOP_START_BRACE "{"
LOOP_OTHERS [^{}\n]*\n
LOOP_END {IF_BODY_END}

NEW_LINE_AND_TAB [\n\t]*

%%

{SINGLE_LINE_COMMENT} { inc(1); initSingleComment(); }
"/*" { initMultiComment(); BEGIN( pushState(COMMENT) ); }


<COMMENT>{SINGLE_LINE_COMMENT} { inc(1); initSingleComment(); }
<MAIN>{SINGLE_LINE_COMMENT} { inc(1); initSingleComment(); }
<VAR_SEC>{SINGLE_LINE_COMMENT} { inc(1); initSingleComment(); }
<IF_SEC>{SINGLE_LINE_COMMENT} { inc(1); initSingleComment(); }
<ELSE_SEC>{SINGLE_LINE_COMMENT} { inc(1); initSingleComment(); }
<IGNORE_SEC>{SINGLE_LINE_COMMENT} { inc(1); initSingleComment(); }
<DISCARD_SEC>{SINGLE_LINE_COMMENT} { inc(1); initSingleComment(); }
<LOOP_SEC>{SINGLE_LINE_COMMENT} { inc(1); initSingleComment(); }
<LOOP_BODY_SEC>{SINGLE_LINE_COMMENT} { inc(1); initSingleComment(); }
<OUT_SEC>{SINGLE_LINE_COMMENT} { inc(1); initSingleComment(); }
<PROTO_SEC>{SINGLE_LINE_COMMENT} { inc(1); initSingleComment(); }
<FUNC_SEC>{SINGLE_LINE_COMMENT} { inc(1); initSingleComment(); }
<FUNC_SEC_VAR>{SINGLE_LINE_COMMENT} { inc(1); initSingleComment(); }

<MAIN>"/*" { initMultiComment(); BEGIN( pushState(COMMENT) ); }
<VAR_SEC>"/*" { initMultiComment(); BEGIN( pushState(COMMENT) ); }
<IF_SEC>"/*" { initMultiComment(); BEGIN( pushState(COMMENT) ); }
<ELSE_SEC>"/*" { initMultiComment(); BEGIN( pushState(COMMENT) ); }
<IGNORE_SEC>"/*" { initMultiComment(); BEGIN( pushState(COMMENT) ); }
<DISCARD_SEC>"/*" { initMultiComment(); BEGIN( pushState(COMMENT) ); }
<LOOP_SEC>"/*" { initMultiComment(); BEGIN( pushState(COMMENT) ); }
<LOOP_BODY_SEC>"/*" { initMultiComment(); BEGIN( pushState(COMMENT) ); }
<OUT_SEC>"/*" { initMultiComment(); BEGIN( pushState(COMMENT) ); }
<PROTO_SEC>"/*" { initMultiComment(); BEGIN( pushState(COMMENT) ); }
<FUNC_SEC>"/*" { initMultiComment(); BEGIN( pushState(COMMENT) ); }
<FUNC_SEC_VAR>"/*" { initMultiComment(); BEGIN( pushState(COMMENT) ); }

<COMMENT>"/*" { appendTextToBuffer("/*"); }

<COMMENT>"*/" {
    appendTextToBuffer("*/"); checkForEnd();
    if(comment_depth == 0){ inc(2); BEGIN( popState() ); }
}
<COMMENT>{NEW_LINE_AND_TAB}* { appendTextToBuffer("|"); }
<COMMENT>. { appendToBuffer(yytext[0]); }

{HEADER} { insertImport(yytext); inc(0); processHeader();}

{FUNC_TYPE} {
    //printf("m_type: -%s-\n",yytext);
    BEGIN( pushState(PROTO_SEC) );
    char *type = removeRedundant(yytext);
    yylval.name =  strdup( type );
    return FUNC_TYPE;
}

<PROTO_SEC>{FUNC_TYPE} {
    //printf("type: -%s-\n",yytext);
    char *type = removeRedundant(yytext);
    yylval.name =  strdup( type );
    return FUNC_TYPE;
}
<PROTO_SEC>{VARIABLE} {
    //printf("name: -%s-\n",yytext);
    yylval.name = strdup(yytext);
    return FUNC_NAME;
}

<PROTO_SEC>"," { return *yytext; }
<PROTO_SEC>"(" { return *yytext; }
<PROTO_SEC>");" { BEGIN( popState() ); return *yytext; }

{MAIN_START} {
    blockBalancer++;
    //printf("main start %s\n",yytext);
    initMain();
    BEGIN( pushState(MAIN) );
    return ENTRY_POINT;
}

<MAIN>{DATA_TYPE} {
    //printf("main data type %s\n",yytext);
    initVarSec(yytext);
    char *type = removeRedundant(yytext);
    yylval.name =  strdup( type );
    BEGIN( pushState(VAR_SEC) );
    return DATA_TYPE;
}

<MAIN>{OUT_START} {
    initOutBuffer();
    BEGIN( pushState(OUT_SEC) );
    return OUTPUT;
}

<MAIN>{IF} {
    inc(7);
    BEGIN( pushState(IF_SEC) );
    return JUST_IN_CASE;
}
<MAIN>{ELSE} {
    inc(10);
    BEGIN( pushState(ELSE_SEC) );
    return ELSE;
}

<MAIN>{LOOP_START} {
    inc(8);
    BEGIN( pushState(LOOP_SEC) );
    return TILL;
}

<MAIN>{FUNC_START} {
    //printf("----%s----",yytext);
    BEGIN( pushState(FUNC_SEC) );
    yylval.name = strdup(yytext);
    return FUNC_NAME;
}

<VAR_SEC>{FUNC_START} {
    //printf("----%s----",yytext);
    BEGIN( pushState(FUNC_SEC_VAR) );
    yylval.name = strdup(yytext);
    return FUNC_NAME;
}

<FUNC_SEC_VAR>("("|",") { return *yytext; }
<FUNC_SEC_VAR>(")") { BEGIN( popState() ); return *yytext; }
<FUNC_SEC_VAR>{VAR_NUM} { yylval.name = strdup(yytext); return VAR_CON; }


<FUNC_SEC>{VAR_NUM} { yylval.name = strdup(yytext); return VAR_CON; }

<FUNC_SEC>("("|")"|",") {
    return *yytext;
}

<FUNC_SEC>";" {
    BEGIN( popState() );
    return *yytext;
}

<MAIN>{VARIABLE} {
    yylval.name = strdup(yytext);
    BEGIN( pushState(VAR_SEC) );
    return VAR;
```

```
}

<VAR_SEC>{NUMBER} {
    sendNumber(yytext);
    return NUMBER;
}
<VAR_SEC>{ARITH_OP} {
    yylval.name = removeLpRp(yytext);
    return ARITH_OPE;
}
<VAR_SEC>{VARIABLE} {
    yylval.name = strdup(yytext);
    return VAR;
}
<VAR_SEC>"," { return *strdup(yytext); }
<VAR_SEC>"=" { return *strdup(yytext); }
<VAR_SEC>";" {
    BEGIN( popState() );
    return *strdup(yytext);
}
<VAR_SEC>{VAR_SPACE} {}


<MAIN>{DISCARD_START} {
    //printf("Discard section\n");
    inc(5);
    BEGIN( pushState(DISCARD_SEC) );
    return DISCARD;
}

<DISCARD_SEC>{VARIABLE} {
    //printf("\nFrom discard only:-%s-\n",yytext);
    yylval.name = strdup(yytext);
    return VAR;
}
<DISCARD_SEC>"," { return *yytext; }
<DISCARD_SEC>";" { BEGIN( popState() ); return *yytext; }


<OUT_SEC>{OUT_BODY_CONST} {
    yylval.name = strdup(yytext);
    return OUTPUT_VC;
}
<OUT_SEC>{OUT_BODY_VAR} {
    yylval.name = strdup(yytext);
    return OUTPUT_VC;
}
<OUT_SEC>{OUT_SEP} {
    return OUTPUT_SEP;
}
<OUT_SEC>{OUT_END} {
    inc(4);
    BEGIN( popState() );
    return OUTPUT_END;
}

<ELSE_SEC>("("|")") { return *yytext; }
<ELSE_SEC>"}" { return END_POINT; }
<ELSE_SEC>{IF_BODY_START} {
    //popState(); // popping else section and taking to main section arki
    popState();

    bool isLastIfValid = getCurrentValidity();

    char *valid = !isLastIfValid ? "True" : "False";
    printf("\nElse condition is %s\n",valid);
    if(isLastIfValid){
        bracketCounter=1;
        BEGIN( pushState(IGNORE_SEC) );
    }
    else{
        BEGIN(pushState(MAIN));
    }
    return *yytext;
}


<IF_SEC>("("|")") { return *yytext; }
<IF_SEC>"}" { return END_POINT; }

<IF_SEC>{COND_OP} { yylval.name = removeLpRp(yytext); return COND_OPE; }
<IF_SEC>"and" { return AND; }
<IF_SEC>"or" { return OR; }
<IF_SEC>{VAR_NUM} { yylval.name = strdup(yytext); return VAR_CON; }
<IF_SEC>{IF_BODY_START} {
    //popState(); // popping if section and taking to main section arki
    popState();

    bool test = getCurrentValidity();
    char *valid = test ? "True" : "False";
    printf("\nIf condition is %s\n",valid);

    if(!test){
        bracketCounter=1;
        BEGIN( pushState(IGNORE_SEC) );
    }
    else{
        BEGIN(pushState(MAIN));
    }
    return *yytext;
}
<IF_SEC>{NEW_LINE_AND_TAB} {}

<IGNORE_SEC>{IGNORE_LEFT_BRACE} { bracketCounter++; }
<IGNORE_SEC>{IGNORE_RIGHT_BRACE} {
    bracketCounter--;
    if(bracketCounter == 0){
        BEGIN( popState() );
        return END_POINT;
    }
}
<IGNORE_SEC>[^}] {}


<LOOP_SEC>("("|")") { return *yytext; }
<LOOP_SEC>"}" { return END_POINT; }
<LOOP_SEC>{COND_OP} { yylval.name = removeLpRp(yytext); return COND_OPE; }
<LOOP_SEC>{VAR_NUM} { yylval.name = strdup(yytext); return VAR_CON; }

<LOOP_SEC>{LOOP_START_BRACE} { BEGIN( pushState(LOOP_BODY_SEC) ); return
*yytext; }

<LOOP_BODY_SEC>{VARIABLE} {
    yylval.name = strdup(yytext);
    BEGIN( pushState(VAR_SEC) );
    return VAR;
}
<LOOP_SEC>{NEW_LINE_AND_TAB} {}
<LOOP_BODY_SEC>{LOOP_END} {
    popState();
    BEGIN( popState() );
    return END_POINT;
}

<LOOP_BODY_SEC>{NEW_LINE_AND_TAB} {}

<MAIN>{MAIN_END} {
    blockBalancer--;
    BEGIN( popState() );
    //printf("\n------main end--\n");
    return END_POINT;
}

<MAIN>{NEW_LINE_AND_TAB}* { }
{NEW_LINE_AND_TAB}* { }

%%

int yywrap(){
    return 1;
}
```