code.l

```
%{
    #include<stdio.h>
    #include<string.h>
    #include <stdlib.h>
    #include <stdbool.h>
    #include <ctype.h>
    #include "constant.h"
    #include "var_list.h"

    // header, single, multi, var, print, dis, calc_val, if, till
    int tempCounter[10];

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

    const int MAXIMUM_VAR_SIZE = 10;

    char error[200];
    char lastDataType[10];

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

    char *outBuffer = NULL;
    int outBufferSize = 0;

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

    size_t buffer_length = 0;

    int bracketCounter=0;

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

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

    bool isLastOneIf(int ifs){
        if(stackSize > 1){
            if( stack[stackSize-1] == ifs){
                return true;
            }
        }
        return false;
    }


    // variable
    const int MAXIMUM_VARIABLE_LENGTH = 200;


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

    char* removeRedundant(char *source){
        int len = strlen(source);

        char temp[len+1];
        int j = 0;
        for(int i=0; i<len; i++){
            //printf("%d, ", source[i]);
            if( source[i] == ' ' ||
                source[i] == ',' ||
                source[i] == ';' ||
                source[i] == '\t' ||
                source[i] == '\n' ||
                source[i] == 32
            )continue;

            temp[j] = source[i];
            j++;
        }

        temp[j] = '\0';
        char *dest;
        dest = malloc(j*sizeof(char));
        strcpy(dest,temp);
        //printf("\n%s\n",dest);
        return dest;
    }

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

    void printAndClearOutBuffer(){
        printf("\n%s < - - - - - - - - - -\n",outBuffer);
        free(outBuffer);
        outBufferSize = 0;
    }

    void continueOutBuffer(char *vc){
        int i=0, len = strlen(vc) - 1;
        while( i <= len && vc[i] == ' ') i++; // leading space
        while( len>=0 && vc[len] == ' ') len--; // trailing space after "
        while( len>=0 && vc[len] == ',' || vc[len]=='\"') len--; // space

        char *res;

        if(vc[i] == '\"'){ // constant
            i++; // trailing quotaion bad dilam

```

```
            int size = len-i+1;
            res = (char *)malloc(size+1);
            strncpy(res, vc + i, size);
            res[size] = '\0';
        }
        else{ // variable
            double val = 0;
            vc = removeRedundant(vc);
            struct VARIABLE *var = getVariable(vc);
            if(var == NULL) {
                printf("-%s-\n",vc);
                stopProgram("Undeclared variable inside println");
            }

            val = var->value;
            res = (char *) malloc(20);
            sprintf(res, "%.3f ", val);
        }

        outBufferSize += strlen(res);
        outBuffer = (char *)realloc(outBuffer,outBufferSize);

        strcat(outBuffer,res);
    }

    void extractNameValue(char *nameVal, char *name, double* val){
        char *temp;
        //printf("Extract name value: %s\n",nameVal);
        temp = removeRedundant(nameVal);
        //printf("--- %s ---\n",temp);
        if( sscanf(temp, "%[^=]=%lf", name, val) == 2){
            free(temp);
            return;
        }

        //printAll();

        char rightVal[20];
        sscanf(temp, "%[^=]=%s", name, rightVal);

        //printf("Right values: -%s-\n",rightVal);

        struct VARIABLE *var = getVariable(rightVal);
        if(var == NULL) {
            printf("%s\n",rightVal);
            stopProgram("Invalid variable in assignment");
        }
        else *val = var->value;

        //printf("values -%s- -%lf-\n",var->name,var->value);

        //printf("Right %s %lf\n",name,*val);
        free(temp);
    }

    bool addVariable(char* name,char *type){
        if( doesVariableExists(name) ){
            stopProgram("Variable is already defined");
            return false;
        }
        insertVariable(name,type,0);
        return true;
    }

    void runHeader(int check){
        if(!check){
            stopProgram("Invalid header format");
        }

        if(!canDeclareHeader){
            printf("Header must be at top of proram\n");
            return;
        }
        else{
            printf("Imported %s\n",yytext);
        }

    }

    void exceptHeader(){
        canDeclareHeader = false;
    }

    void initSingleComment(){
        exceptHeader();
        commentCounter++;
    }

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

    void process_comment() {
        commentCounter++;
        //printf("Captured comment: %s\n", comment_buffer);
    }

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

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

        memset(comment_buffer, 0, 3);

        strcat(comment_buffer, "/*");
    }

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

    void stopMain(){
        printf("Execution done\n");
        isMainBlockFound = true;
    }

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

    void assignValue(char *name, double val){
        updateVariable(name,val);
    }

    void removeVariable(char *name){
        name = removeRedundant(name);
        deleteVariable(name);
    }

    void declareVariable(char *name, double val){
        name = removeRedundant(name);

        //printf("\n declaring.................\n");
        //printAll();
        if( !addVariable(name, removeRedundant(lastDataType) ) ){
            char message[50];
            strcpy(message,"Duplicate variable found ");
            strcat(message,name);
            stopProgram(message);
            return;
        }
        //printf("\nBefore assign -%s- -%lf-\n",name,val);
        assignValue(name,val);
        //printAll();
    }

    void processVariable(char *nameVal){
        //printf("nameVal %s\n",nameVal);

        char name[20];
        double value;
        extractNameValue(nameVal,name,&value);
        //printf("\nname -%s- -%lf-\n",name,value);
        declareVariable(name,value);
    }

    void continueVariableCalc(char *stmt, bool addVar){

        //updateValue(yytext);
        char leftVar[30], varBefOp[30], op[10], varAftOp[30];
        processStatement(stmt, leftVar, varBefOp, op, varAftOp);

        if(addVar){
            declareVariable(leftVar,0);
        }

        double temp;
        if( sscanf(leftVar, "%lf", &temp) == 1){
            stopProgram("\nCan't assign value to constant.");
        }

        if( !doesVariableExists(leftVar) ){
            stopProgram("\nVariable not declared before");
        }

        double befVal;
        if( sscanf(varBefOp, "%lf", &befVal) != 1){
            struct VARIABLE *val = getVariable(varBefOp);
            if(val == NULL) {
                stopProgram("\nInvalid variable found.");
            }
            else befVal = (*val).value;
        }

        double aftVal;
        if( sscanf(varAftOp, "%lf", &aftVal) != 1){
            struct VARIABLE *val = getVariable(varAftOp);
            if(val == NULL) {
                stopProgram("\nInvalid variable found.");
            }
            else aftVal = (*val).value;
        }

        //printf("\nbefore %lf, after %lf\n",befVal,aftVal);

        //printf("\ncalc val: %lf",getValue(befVal,aftVal,op) );

        assignValue( leftVar, getValue(befVal,aftVal,op) );
    }

    void updateValue(char *nameVal){
        //printf("Passed %s\n",nameVal);
        char name[20];
        double val;
        extractNameValue(nameVal,name,&val);
        //printf("Extracted %s %lf\n",name,val);

        if( !doesVariableExists(name) ){
            char message[50];
            strcpy(message, "Variable doesn't exists: ");
            strcat(message,name);

            stopProgram(message);
            return;
```

```
        }
        //printf("Value not assigned\n");
        assignValue(name,val);
        //printf("Value assigned\n");
    }

    void discardVariable(char *name){
        name = removeRedundant(name);
        if( !doesVariableExists(name) ){
            printf("\nCan't discard %s since not found. Ignoring...\n",name);
            return;
        }
        deleteVariable(name);
    }

    bool isNumber(char* num){
        bool count = 0;
        for(int i=0; i<strlen(num); i++){
            if( num[i] == '.' && count == 0 ) { count=1; continue; }
            if( !isdigit(num[i]) ) return false;
        }
        return true;
    }

    void processIfBody(char *body){
        //processed body
        char pBody[strlen(body)];

        int i=0;
        while(body[i] != '(')i++;
        i++;

        int j=0;
        while(body[i] != ')'){ pBody[j] = body[i]; i++; j++; }
        pBody[j] = '\0';

        char leftVal[50], operator[10], rightVal[50];
        sscanf(pBody, "%49s %9s %49s", leftVal, operator, rightVal);

        double left = 0, right = 0;

        if( isNumber(leftVal) ) left = strtod(leftVal,NULL);
        else{
            struct VARIABLE *val = getVariable(leftVal);
            if(val == NULL) {
                printf("%s\n",leftVal);
                stopProgram("Invalid variable in if");
            }
            else left = (*val).value;
        }

        if( isNumber(rightVal) ) right = strtod(rightVal,NULL);
        else{
            struct VARIABLE *val = getVariable(rightVal);
            if(val == NULL) {
                printf("%s\n",rightVal);
                stopProgram("Invalid variable in if");
            }
            else right = (*val).value;
        }

        if( !isCondOpValid(operator) ){
            stopProgram("Invalid operator in if");
            return;
        }

        isLastIfValid = isConditionValid(left,operator,right);

        //printf("body of if is: %d\n",isLastIfValid);

        //printf("extracted: %s %s %s\n",leftVar,operator,rightVal);
    }

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


%}
```