Practical - 3

Aim : Implement Lexical Analyzer using C Language. (The lexical analyzer should ignore redundant spaces, tabs and new lines. It should also ignore comments. Simulate the same in C language.)

Program:

lexical_analyzer.c :

```
/*****************************************************************
        Author : Bhishm Daslaniya [17CE023]
        "Make it work, make it right, make it fast."
                                                – Kent Beck
*****************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#define FALSE 0
#define TRUE 1

FILE * source;
FILE * listing;
FILE * code;
int EchoSource = FALSE;
int TraceScan = TRUE;
int Error = FALSE;

typedef enum{
        /* reserved words */
        RESERVEDWORD,
        /* set */
        ID,NUM,
        /* simple symbols */
        PLUS,MINUS,MUL,LPAREN,RPAREN,SEMI,COMMA,XOR,
        LSQUARE,RSQUARE,LBRACE,RBRACE,MOD,DOT,
        LOGICAL_AND,LOGICAL_OR,
        /* diffcult symbols*/
        LE,GE,NEQ,NOT,ASSIGN,EQ,LT,GT,DIV,BITWISE_OR,BITWISE_AND,
        LEFT_SHIFT,RIGHT_SHIFT,INCR,DECR,ARROW,SINGLE_QUOTE,
        PLUS_ASSIGN,MINUS_ASSIGN,MUL_ASSIGN,DIV_ASSIGN,MOD_ASSIGN,
        /*comment*/
        COMMENT,
        /*other*/
        PREPROCESSOR,STR_LITERAL,ENDFILE,ERROR
}TokenType;

typedef enum{
        START,
        ASSIGNSTATE,
        ENTERCOMMENT,COMMENTSTATE,OUTCOMMENT,
```

```
        ENTERSTR,OUTSTR,
        NUMSTATE,IDSTATE,
        LESTATE,
        GESTATE,
        NEQSTATE,
        SINGLE_QUOTE_STATE,BITWISE_AND_STATE,BITWISE_OR_STATE,
        PLUS_STATE,MINUS_STATE,MUL_STATE,MOD_STATE,REAL_VAL_STATE,
        OVER
}StateType;

static char reservedWords[32][10] = {
        "auto","break","case","char","const","continue","default","do",
        "double","else","enum","extern","float","for","goto","if",
        "int","long","register","return","short","signed","sizeof","static",
        "struct","switch","typedef","union","unsigned","void","volatile","while"
};

// Main functions start for lexical analyzer...

// Functions and variacles declarations
void printToken(TokenType, const char*);
int getNextChar(void);
TokenType checkReserved (char* s);

static int pos=0;
static int lineid=0;
static char currentLine[500];//set size 500
static int currentLineSize;
static int isEnd=FALSE;
static int isComment=FALSE;
char getString[500];

TokenType getToken(void){
        StateType state = START;
        TokenType currentToken=ID;//Initialization
        int getStringSize=0;
        int count_char = 0;

        while (state!=OVER){
                int isSave=TRUE;
                int c=getNextChar();
                // printf("%c  sss\n",c);
                if(isComment){
                        isSave=FALSE;
                        state=OVER;
                        return TokenType(1);
                }
                switch (state){
                        case START:
                                //diffcult and set
                                if(isalpha(c))
                                        state=IDSTATE;
```

```
else if(isdigit(c))
        state=NUMSTATE;
else if(c=='<')
        state=LESTATE;
else if(c=='>')
        state=GESTATE;
else if(c=='=')
        state=ASSIGNSTATE;
else if(c=='!')
        state=NEQSTATE;
else if(c=='/')
        state=ENTERCOMMENT;
else if(c=='\"')
        state=ENTERSTR;
else if(c=='\'')
        state=SINGLE_QUOTE_STATE;
else if(c=='&')
        state = BITWISE_AND_STATE;
else if(c=='|')
        state = BITWISE_OR_STATE;
else if(c=='+')
        state = PLUS_STATE;
else if(c=='-')
        state= MINUS_STATE;
else if(c=='*')
        state = MUL_STATE;
else if(c==37)
        state = MOD_STATE;
else if(c=='.')
        state = REAL_VAL_STATE;
//blank
else if ((c == ' ') || (c == '\t') || (c == '\n'))
        isSave = FALSE;
//simple
else{
        state=OVER;
        switch (c){
                case 0:
                        currentToken=ENDFILE;
                        break;
                case '^':
                        currentToken = XOR;
                        break;
                case '(':
                        currentToken = LPAREN;
                        break;
                case ')':
                        currentToken = RPAREN;
                        break;
                case '[':
                        currentToken=LSQUARE;
                        break;
```

```
                        case ']':
                                currentToken=RSQUARE;
                                break;
                        case '{':
                                currentToken=LBRACE;
                                break;
                        case '}':
                                currentToken=RBRACE;
                                break;
                        case ';':
                                currentToken = SEMI;
                                break;
                        case ',':
                                currentToken=COMMA;
                                break;
                        default:
                                currentToken = ERROR;
                                break;
                }
        }
        break;
//state ananlyze
case IDSTATE:
        if(!(isalpha(c)||isdigit(c))){
                pos--;//reset pos to previous char for next analyze
                isSave=FALSE;//it is not this token's char
                state=OVER;
                currentToken=ID;
        }
        break;
case NUMSTATE:
        if(c=='.'){
                state = REAL_VAL_STATE;
                isSave = TRUE;
        }else if(!isdigit(c)){
        pos--;
        isSave=FALSE;
        state=OVER;
        currentToken=NUM;
        }
        break;
case PLUS_STATE:
        if(c=='+'){
                state=OVER;
                currentToken = INCR;
        }else if(c=='='){
                state = OVER;
                currentToken = PLUS_ASSIGN;
        }else{
                pos--;
                isSave = FALSE;
                state = OVER;
```

```
                currentToken = PLUS;
        }
        break;
case MINUS_STATE:
        if(c=='-'){
                state=OVER;
                currentToken = DECR;
        }else if(c=='>'){
                state = OVER;
                currentToken = ARROW;
        }else if(c=='='){
                state = OVER;
                currentToken = MINUS_ASSIGN;
        }else{
                pos--;
                isSave = FALSE;
                state = OVER;
                currentToken = MINUS;
        }
        break;
case MUL_STATE:
        if(c=='='){
                state = OVER;
                currentToken = MUL_ASSIGN;
        }else{
                pos--;
                isSave = FALSE;
                state = OVER;
                currentToken = MUL;
        }
        break;
case MOD_STATE:
        if(c=='='){
                state = OVER;
                currentToken = MOD_ASSIGN;
        }else{
                pos--;
                isSave = FALSE;
                state = OVER;
                currentToken = MOD;
        }
        break;
case LESTATE:
        if(c=='='){
                state=OVER;
                currentToken=LE;
        }else if(c=='<'){
                state = OVER;
                currentToken = LEFT_SHIFT;
        }else{
                pos--;
                isSave=FALSE;
```

```
                    state=OVER;
                    currentToken=LT;
            }
            break;
    case GESTATE:
            if(c=='='){
                    state=OVER;
                    currentToken=GE;
            }else if(c=='>'){
                    state = OVER;
                    currentToken = RIGHT_SHIFT;
            }else{
                    pos--;
                    isSave=FALSE;
                    state=OVER;
                    currentToken=GT;
            }
            break;
    case ASSIGNSTATE:
            if(c=='='){
                    state=OVER;
                    currentToken=EQ;
            }else{
                    pos--;
                    isSave=FALSE;
                    state=OVER;
                    currentToken=ASSIGN;
            }
            break;
    case REAL_VAL_STATE:
            if(isdigit(c)){
                    state = NUMSTATE;
            }else{
                    pos--;
                    state = OVER;
                    isSave = FALSE;
                    currentToken = DOT;
            }
            break;
    case NEQSTATE:
       if(c=='='){
                    state=OVER;
                    currentToken=NEQ;
            }else{
                    pos--;
                    isSave=FALSE;
                    state=OVER;
                    currentToken=NOT;
            }
            break;
    case BITWISE_OR_STATE:
            if(c=='|'){
```

```
                state = OVER;
                currentToken = LOGICAL_OR;
        }else{
                pos--;
                isSave=FALSE;
                state = OVER;
                currentToken = BITWISE_OR;
        }
        break;
case BITWISE_AND_STATE:
        if(c=='&'){
                state = OVER;
                currentToken = LOGICAL_AND;
        }else{
                pos--;
                isSave=FALSE;
                state = OVER;
                currentToken = BITWISE_AND;
        }
        break;
case SINGLE_QUOTE_STATE:
        if(c!='\''){
                state = SINGLE_QUOTE_STATE;
                isSave = TRUE;
                count_char++;
        }else if(c=='\'' && count_char <= 1){
                state = OVER;
                currentToken = SINGLE_QUOTE;
        }else if(c=='\'' && count_char > 1){
                state = OVER;
                currentToken = ERROR;
        }else{
                if(isEnd){
                        state = OVER;
                        currentToken = ERROR;
                }
        }
        break;
case ENTERCOMMENT:
        if(c=='/'){
                state=OVER;
                currentToken=COMMENT;
                isComment=TRUE;
        }else if(c=='*'){
                state=COMMENTSTATE;
        }else if(c=='='){
                state = OVER;
                currentToken = DIV_ASSIGN;
        }else{
                pos--;
                isSave=FALSE;
                state=OVER;
```

```
                        currentToken=DIV;
                }
                break;
        case COMMENTSTATE:
                if(c=='*'){
                        state=OUTCOMMENT;
                }else{
                        state=COMMENTSTATE;
                        if(isEnd){
                                state=OVER;
                                currentToken=ERROR;
                        }
                }
                break;
        case OUTCOMMENT:
                if(c=='/'){
                        state=OVER;
                        currentToken=COMMENT;
                }else{
                        state=COMMENTSTATE;
                        if(isEnd){
                                state=OVER;
                                currentToken=ERROR;
                        }
                }
                break;
        case ENTERSTR:
                if(c!='\'"){
                        state = ENTERSTR;
                        isSave = TRUE;
                }else if(c == '\'"){
                        state = OVER;
                        currentToken = STR_LITERAL;
                }else{
                        if(isEnd){
                                state = OVER;
                                currentToken = ERROR;
                        }
                }
                break;
        case OVER:
        default:
                printf("ERROR IN %d\n",state);
                state=OVER;
                currentToken=ERROR;
                break;
}
if(isSave)
        getString[getStringSize++]=(char)c;
if(state==OVER){
        getString[getStringSize]='\0';
}
```

```
        }
        printf("line:%d    ",lineid);//print line id first
        if(currentToken==ID){
                currentToken=checkReserved(getString);
        }
        printToken(currentToken,getString);
        return currentToken;
}

int validIdentifier(const char* str){
        if (!((str[0] >= 'a' && str[0] <= 'z')
       || (str[0] >= 'A' && str[1] <= 'Z')
       || str[0] == '_'))
      return FALSE;

   // Traverse the string for the rest of the characters
   for (int i = 1; i < strlen(str); i++) {
      if (!((str[i] >= 'a' && str[i] <= 'z')
         || (str[i] >= 'A' && str[i] <= 'Z')
         || (str[i] >= '0' && str[i] <= '9')
         || str[i] == '_'))
        return FALSE;
   }
        return TRUE;
}

void printToken(TokenType token,const char* getString){
        switch(token){
                case MOD:
                        printf("Operator:                       %%\n");
                        break;
                case ASSIGN:
                        printf("Operator:                       =\n");
                        break;
                case LT:
                        printf("Operator:                       <\n");
                        break;
                case LE:
                        printf("Operator:                       <=\n");
                        break;
                case GT:
                        printf("Operator:                       >\n");
                        break;
                case GE:
                        printf("Operator:                       >=\n");
                        break;
                case EQ:
                        printf("Operator:                       ==\n");
                        break;
                case NEQ:
                        printf("Operator:                       !=\n");
                        break;
```

```
case NOT:
        printf("Operator:           !\n");
        break;
case LPAREN:
        printf("Special Symbol:     (\n");
        break;
case RPAREN:
        printf("Special Symbol:     )\n");
        break;
case SEMI:
        printf("Special Symbol:     ;\n");
        break;
case COMMA:
        printf("Special Symbol:     ,\n");
        break;
case LSQUARE:
        printf("Special Symbol:     [\n");
        break;
case RSQUARE:
        printf("Special Symbol:     ]\n");
        break;
case LBRACE:
        printf("Special Symbol:     {\n");
        break;
case RBRACE:
        printf("Special Symbol:     }\n");
        break;
case DOT:
        printf("Special Symbol:     .\n");
        break;
case PLUS:
        printf("Operator:           +\n");
        break;
case MINUS:
        printf("Operator:           -\n");
        break;
case XOR:
        printf("Operator:           ^\n");
        break;
case MUL:
        printf("Operator:           *\n");
        break;
case DIV:
        printf("Operator:           /\n");
        break;
case BITWISE_OR:
        printf("Operator:           |\n");
        break;
case BITWISE_AND:
        printf("Operator:           &\n");
        break;
case LOGICAL_AND:
```

```
            printf("Operator:                   &&\n");
            break;
        case LOGICAL_OR:
            printf("Operator:                   ||\n");
            break;
        case ARROW:
            printf("Operator:                   ->\n");
            break;
        case DIV_ASSIGN:
            printf("Operator:                   /=\n");
            break;
        case MUL_ASSIGN:
            printf("Operator:                   *=\n");
            break;
        case PLUS_ASSIGN:
            printf("Operator:                   +=\n");
            break;
        case MINUS_ASSIGN:
            printf("Operator:                   -=\n");
            break;
        case MOD_ASSIGN:
            printf("Operator:                   %%=\n");
            break;
        case LEFT_SHIFT:
            printf("Operator:                   <<\n");
            break;
        case RIGHT_SHIFT:
            printf("Operator:                   >>\n");
            break;
        case INCR:
            printf("Operator:                   ++\n");
            break;
        case DECR:
            printf("Operator:                   --\n");
            break;
        case SINGLE_QUOTE:
            printf("Single character:           %s\n",getString);
            break;
        case ENDFILE:
            printf("END\n");
            break;
        case COMMENT:
            printf("COMMENT\n");
            break;
        case NUM:
            printf("Number:                     Value = %s\n",getString);
            break;
        case ID:
            if(validIdentifier(getString)){
                printf("Identifier:                 idName = %s\n",getString);
            }else{
                printf("<<<ERROR>>> Identifier not valid: idName = %s\n",getString);
```

```
                }
                break;
            case RESERVEDWORD:
                printf("Reserved Keyword:       %s\n",getString);
                break;
            case STR_LITERAL:
                printf("String literal:         %s\n",getString);
                break;
            case ERROR:
                printf("<<<ERROR>>>             in this line\n");
                break;
            default: /* should never happen */
                printf("Unknown token:      %d\n",token);
                break;
        }
}

int getNextChar(void){
        if(pos>=currentLineSize){
            isComment=FALSE;
            lineid++;
            if (fgets(currentLine,500,source)){
                currentLineSize = strlen(currentLine);
                pos = 0;
                return currentLine[pos++];
            }else{
                isEnd = TRUE;
                return 0;
            }
        }else
            return currentLine[pos++];
}

TokenType checkReserved (char* s){
        int i;
        for(i=0;i<32;i++){
            if(!strcmp(s,reservedWords[i])){
                return RESERVEDWORD;
            }
        }
        return ID;
}
int main(int argc, char * argv[]){
        char sourceFile[200];
        strcpy(sourceFile,argv[1]) ;
        source = fopen(sourceFile,"r");
        printf("lexical analyze-> %s\n",sourceFile);
        printf("-------------------------------------------------------\n");
        while (getToken()!=ENDFILE);
        fclose(source);
        return 0;
}
```

test_input.c:

```
// Input without preprocessor directives...
/*
	Author : Bhishm Daslaniya [17CE023]
	"Make it work, make it right, make it fast."
                                                – Kent Beck
*/

int main(){
	int sum = 1;
	for(int i = 0 ; i < 10; i+=1){
		sum = sum<<i;
		printf("Iteration %d\n",i);
	}
	scanf("sum : %d",&sum);
	float f = 1.2222;
	char c = 'c';
	char ch ='cdcd';
	char str[100] = "Copyright by Bhishm Daslaniya";
	int 1b = 55;
	++sum;
	return 0;
}
```

Output:

```
bhishm@BhishmDaslaniya:/media/bhishm/Projects/DLP_lab/Practical_3$ g++ lexical_a
nalyzer.c
bhishm@BhishmDaslaniya:/media/bhishm/Projects/DLP_lab/Practical_3$ ./a.out test_
input.c
lexical analyze-> test_input.c
------------------------------------------------------------
line:1    COMMENT
line:6    COMMENT
line:8    Reserved Keyword:             int
line:8    Identifier:                   idName = main
line:8    Special Symbol:               (
line:8    Special Symbol:               )
line:8    Special Symbol:               {
line:9    Reserved Keyword:             int
line:9    Identifier:                   idName = sum
line:9    Operator:                     =
line:9    Number:                       Value = 1
line:9    Special Symbol:               ;
line:10    Reserved Keyword:            for
line:10    Special Symbol:              (
line:10    Reserved Keyword:            int
line:10    Identifier:                  idName = i
line:10    Operator:                    =
line:10    Number:                      Value = 0
line:10    Special Symbol:              ;
line:10    Identifier:                  idName = i
line:10    Operator:                    <
line:10    Number:                      Value = 10
line:10    Special Symbol:              ;
line:10    Identifier:                  idName = i
line:10    Operator:                    +=
line:10    Number:                      Value = 1
line:10    Special Symbol:              )
line:10    Special Symbol:              {
line:11    Identifier:                  idName = sum
line:11    Operator:                    =
line:11    Identifier:                  idName = sum
line:11    Operator:                    <<
line:11    Identifier:                  idName = i
line:11    Special Symbol:              ;
line:12    Identifier:                  idName = printf
line:12    Special Symbol:              (
line:12    String literal:              "Iteration %d\n"
```

```
line:12    String literal:              "Iteration %d\n"
line:12    Special Symbol:              ,
line:12    Identifier:                  idName = i
line:12    Special Symbol:              )
line:12    Special Symbol:              ;
line:13    Special Symbol:              }
line:14    Identifier:                  idName = scanf
line:14    Special Symbol:              (
line:14    String literal:              "sum : %d"
line:14    Special Symbol:              ,
line:14    Operator:                    &
line:14    Identifier:                  idName = sum
line:14    Special Symbol:              )
line:14    Special Symbol:              ;
line:15    Reserved Keyword:            float
line:15    Identifier:                  idName = f
line:15    Operator:                    =
line:15    Number:                      Value = 1.2222
line:15    Special Symbol:              ;
line:16    Reserved Keyword:            char
line:16    Identifier:                  idName = c
line:16    Operator:                    =
line:16    Single character:            'c'
line:16    Special Symbol:              ;
line:17    Reserved Keyword:            char
line:17    Identifier:                  idName = ch
line:17    Operator:                    =
line:17    <<<ERROR>>>          in this line
line:17    Special Symbol:              ;
line:18    Reserved Keyword:            char
line:18    Identifier:                  idName = str
line:18    Special Symbol:              [
line:18    Number:                      Value = 100
line:18    Special Symbol:              ]
line:18    Operator:                    =
line:18    String literal:              "Copyright by Bhishm Daslaniya"
line:18    Special Symbol:              ;
line:19    Reserved Keyword:            int
line:19    Number:                      Value = 1
line:19    Identifier:                  idName = b
line:19    Operator:                    =
line:19    Number:                      Value = 55
line:19    Special Symbol:              ;
line:20    Operator:                    ++
line:20    Identifier:                  idName = sum
line:20    Special Symbol:              ;
line:21    Reserved Keyword:            return
line:21    Number:                      Value = 0
line:21    Special Symbol:              ;
line:22    Special Symbol:              }
line:23    END
```

Conclusion: From this practical I have learnt about how actual lexical analyzer works and how to implement it for any programming languages.