SRM UNIVERSITY AP
Andhra Pradesh

Department of Computer Science and Engineering
Compiler Design Lab (CS 306)

Week 2: Implementation of Lexical Analyser using C

Week 2 Program

1. Implement lexical analyser using C for recognizing the following tokens:
    - 24 keywords (given in the following program)
    - Identifiers with the regular expression : letter(letter | digit)*
    - Integers with the regular expression: digit+
    - Relational operators: <, >, <=, >=, ==, !=
    - Ignores everything between multi line comments (/* …. */)
    - Storing identifiers in symbol table.

Instructions:

- Explanation and code of the program with the following features are given below.
    - 24 keywords (given in the following program)
    - Identifiers with the regular expression : letter(letter | digit)*
    - Integers with the regular expression: digit+
    - Relational operators: <, >, <=, >=, ==, !=
    - Ignores everything between multi line comments (/* …. */)
- Modify the program to include the code for
    - Storing identifiers in symbol table
    - Printing the message for identification of specific relational operator instead of a general identifier.
- You can optionally add more lexeme recognition by writing proper definitions and assumptions.
- YouTube link of this week's explanation is https://youtu.be/1_WZghls6dw
- Upload the modified programs into your Github accounts under the folder **Week2-Lab-exercise**

Description:

- The program reads input from a text file "x.txt" and writes output into a file "y.txt"
- The implementation of lexical analyser is inspired by Finite Automata. The system is assumed to be in state 0 initially.
- **Recognition of Keywords:** A list of keywords is stored in an array and compared with the read word whether it matches any keyword in the list.
- **Recognition of keywords and identifier**:
    - **Reading input**: When FA reads a character(letter), it enters state 1 and starts reading further characters (letters or digits). On reading a character other than letter and digit, FA goes back to start state by moving the read

character back into stdin to facilitate recognition of next lexeme. Store the read string in a character array.

- **Differentiate keyword/identifier**: A list of keywords is stored in an array and compared with the read string whether it matches any keyword in the list. If matches, it is a keyword. Otherwise, it is identifier.

- **Recognition of operators**: Being in state 0, if FA reads either '<', or '>', , it enters state 5. Further checks whether an '=' follows. Recognize an operator lexeme. Similar is the case with '=' or '!'.
- **Recognition of comments:** When FA reads '/', it goes to state 10, then reads a '*' and enters state 11. Later, FA reads and ignores everything till it encounters '* followed by a '/'. Lexical analyser reports an error message if there is missing close of comments or goes back to state 0 for reading next lexeme.
- Fig.1 describes the overall picture of the lexical analyser.

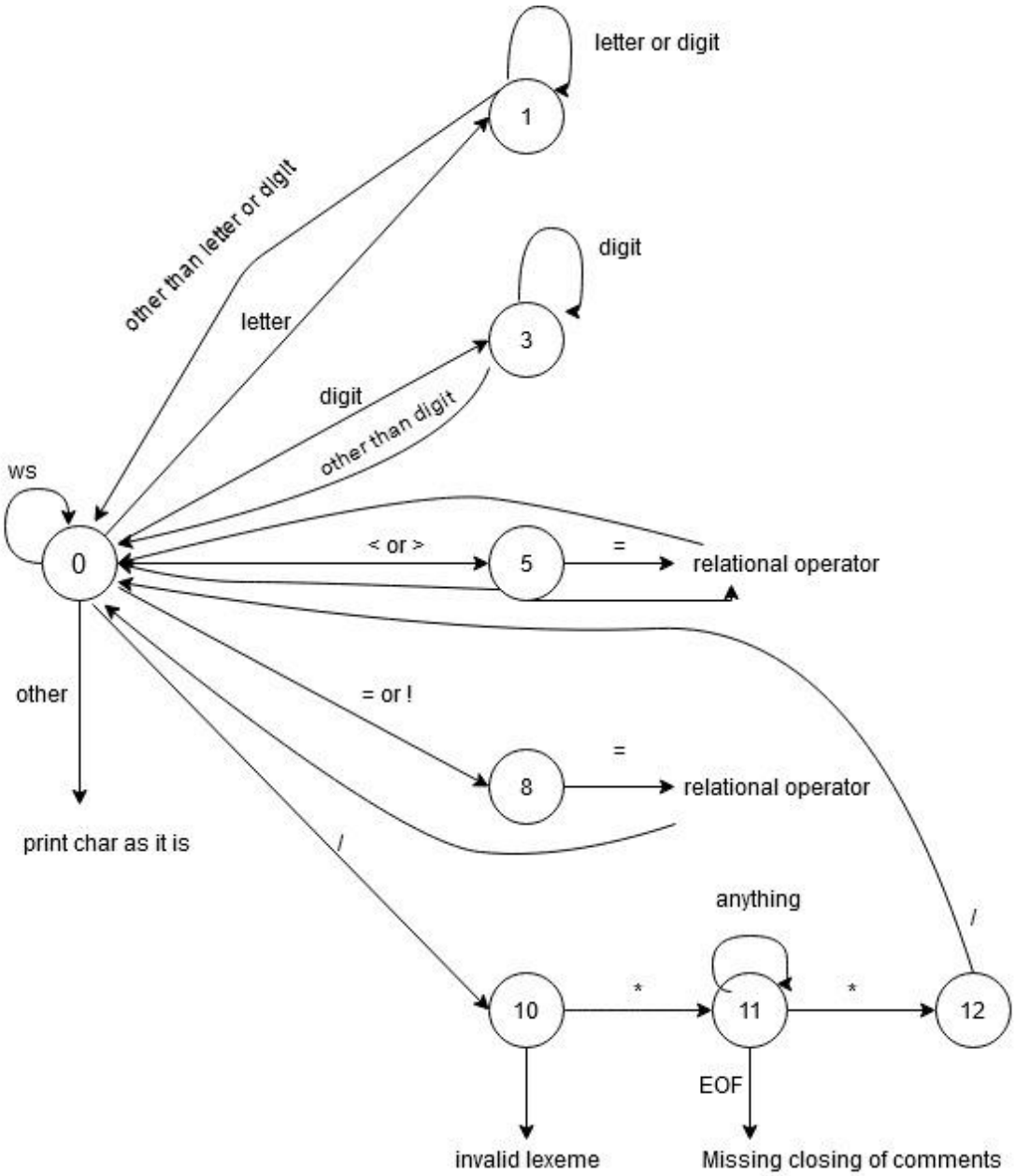

Figure 1: Finite Automata depicting Lexical Analyser

Code

```
/*C program for lexical analyser:
Keywords:
Identifier:
Number : Integers
Relational Operators: <, <=, >, >=, ==, !=
Multi line Comments:
*/

#include<stdio.h>
#include<ctype.h>
#include<string.h>
char
keyword[30][30]={"int","while","break","for","do","if","float","char","switch","d
ouble","short","long","unsigned","sizeof","else","register","extern","static","auto
","case","break","volatile","enum","typedef"};
char id[20], num[10];

//declare symbol table as a doubly dimensional array of characters.

int check_keyword(char s[])
        {
        int i;
        for(i=0;i<24;i++)
          if(strcmp(s,keyword[i])==0)
                return 1;
        return 0;
        }

/*write a function to store identifier in symbol table
void store_symb_tab(char id[], char symb_tab[][20])
{
Check whether the id is already available in the symbol table, if available,
ignore. otherwise add it.
}
*/

int main()
 {
   FILE *fp1,*fp2;
   char c;
   int state=0;
   int i=0,j=0;
   fp1=fopen("x.txt","r");//input file containing src prog
   fp2=fopen("y.txt","w");//output file name
```

```
while((c=fgetc(fp1))!=EOF)
{
   switch(state)
   {
      case 0: if(isalpha(c)){
                 state=1; id[i++]=c;}
              else if(isdigit(c)){
                    state=3; num[j++]=c;}
                 else if(c=='<' || c=='>')
                       state=5;
                    else if(c=='=' || c=='!')
                          state=8;
                       else if(c=='/')
                             state=10;
                          else if(c==' ' || c=='\t' || c=='\n')
                                state=0;
                             else
                                  fprintf(fp2,"\n%c",c);
             break;
      case 1:if(isalnum(c)){
              state=1; id[i++]=c;
             }
             else{
                 id[i]='\0';
                 if(check_keyword(id))
                    fprintf(fp2," \n %s : keyword ",id);
                 else
                    fprintf(fp2,"\n %s : identifier",id);

                    // call a function which stores id in symbol table

                 state=0;
                 i=0;
                 ungetc(c,fp1);
             }
        break;
     case 3:if(isdigit(c)){
              num[j++]=c;
              state=3;
            }
            else{
              num[j]='\0';
              fprintf(fp2," \n%s: number",num);
              state=0;
              j=0;
              ungetc(c,fp1);
```

```
    }
    break;
case 5:if(c=='='){
        fprintf(fp2,"\n  relational operator ");
        //write code to print specific operator like <= or >=
        state=0;
    }
    else{
        fprintf(fp2,"\n  relational operator ");

        //write code to print specific operator like <, >, <= or >=

        state=0;
        ungetc(c,fp1);
    }
    break;
case 8:if(c=='='){
        fprintf(fp2,"\n relational operator ");

        //write code to print specific operator like == or !=

        state=0;

    }
    else{
        ungetc(c,fp1);
        state=0;
    }
    break;
case 10:if(c=='*')
        state=11;
    else
        fprintf(fp2,"\n invalid lexeme");
    break;
case 11: if(c=='*')
        state=12;
    else
        state=11;
    break;
case 12:if(c=='*')
        state=12;
      else if(c=='/')
           state=0;
          else
           state=11;
      break;
```

```
        }//End of switch
      }//end of while
      if(state==11)
       fprintf(fp2,"comment did not close");
      fclose(fp1);
      fclose(fp2);
 return 0;
 }
```

Test cases:

You need to provide source program containing at least one number, keyword, identifier, relational operator and comment, invalid lexemes and missing comments.