Sharvan Ram Kumaran
Reg No:185001143
CSE C
Exercise 1
Objective:
Develop a Lexical analyzer to recognize the patterns namely, identifiers, constants, comments and operators using the following regular expressions.

Code:
```
// Lexical analyser - scans code and recognizes tokens

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <fcntl.h>
#include <stdbool.h>

int isOperator(char ch){

  if (ch == '+' || ch == '-' || ch == '*' || ch == '/' || ch == '%'){
      return 1;
    }
  else if (ch == '>' || ch == '<'){
    return 2;
  }
  else if(ch == '|' || ch == '&'){
    return 3;
  }
  else if(ch == '='){
    return 4;
  }
  return 0;
}

bool isKeyword(char *str){

  if(!strcmp(str, "if") || !strcmp(str, "else") || !strcmp(str, "while") ||
    !strcmp(str, "for") || !strcmp(str, "do") || !strcmp(str, "break") ||
    !strcmp(str, "switch") || !strcmp(str, "continue") || !strcmp(str, "return") ||
    !strcmp(str, "case") || !strcmp(str, "default") || !strcmp(str, "void") ||
    !strcmp(str, "int") || !strcmp(str, "char") || !strcmp(str, "bool") ||
    !strcmp(str, "struct") || !strcmp(str, "goto") || !strcmp(str, "typedef") ||
        !strcmp(str, "unsigned") || !strcmp(str, "long") || !strcmp(str, "short") ||
        !strcmp(str, "float") || !strcmp(str, "double") || !strcmp(str, "sizeof")){
          return true;
        }

    return false;
}
bool isSeparator(char ch){
  if(ch=='{' || ch=='}' || ch==';' || ch=='(' || ch==')' || ch==','){
    return true;
  }
  return false;
}
bool isFunc(char *str){
  if(strcmp(str,"main")==0 || strcmp(str,"printf")==0 || strcmp(str,"scanf")==0)
            {
                        return true;
            }
  return false;
}

void lexanalyse(char *input){
  int i=0,j=0;
            char ch,str[100];
  for(i=0;i<strlen(input);i++){
    ch = input[i];
    if(ch=='#'){
      printf("PDIR ");
      while(input[i]!='\n'){
        i++;
      }
    }
    if(ch=='/'){
      if(input[i+1]=='/'){
        printf("SNGLINE ");
        i+=2;
        while(input[i]!='\n'){
          i++;
        }
      }
      else if(input[i+1]=='*'){
        i+=2;
        printf("MLTLINE ");
        while(input[i]!='*' && input[i+1]!='/'){
```

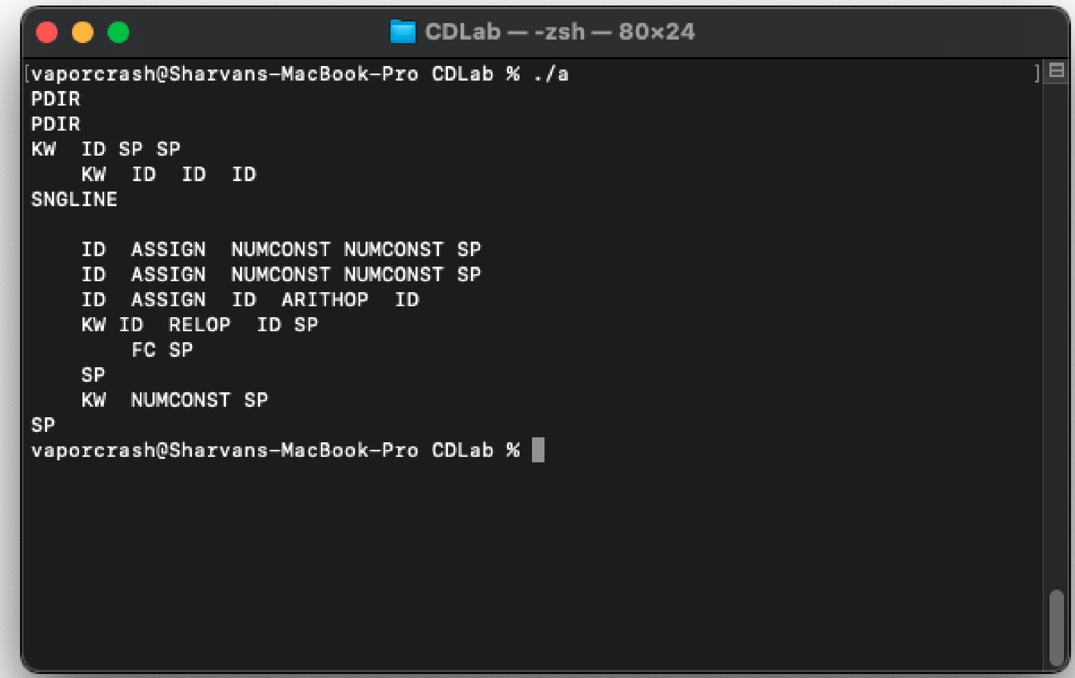
```
      i++;
    }
  }

}
int op = isOperator(ch);
if(op==4){
  ch = input[++i];
  if(ch=='=' || ch=='!'){
    printf("RELOP ");
  }
  else if(ch==' '){
    printf("ASSIGN ");
  }
}
else if(op==2){
  ch = input[++i];
  if(ch=='=' || ch == ' ' || ch == '!'){
    printf("RELOP ");
  }
}
else if(op==3){
  if(ch == input[i+1]){
    printf("LOGICALOP ");
  }
}
else if(op==1){
  ch = input[++i];
  if(ch=='=' || ch == '!'){
    printf("ASSIGN ");
  }
  else if(ch==' '){
    printf("ARITHOP ");
  }

}


if(isSeparator(ch)){
  printf("SP ");
}
if(isalnum(ch)){
  if(isalpha(ch)){
    while(isalnum(ch)){
          str[j++]=ch;
          ch=input[++i];
        }
        str[j]='\0';
        if(isFunc(str)){
          printf("FC ");
          while(input[i]!=')'){
            i++;
          }
        }
        else if(isKeyword(str)){
          printf("KW ");
        }
        else{
          printf("ID ");
        }
      }
      else{
        printf("NUMCONST ");
      }
    }
    if(ch==' '){
      printf(" ");
    }
  }
}

int main(){
  FILE *fp;
  char input[100];
  fp = fopen("sample.c","r");
  while(fgets(input,100,fp)){
    lexanalyse(input);
    printf("\n");
  }
  fclose(fp);
}
```

Sample File:

```
#include<stdio.h>
#include<stdlib.h>
int main(){
    int a, b, c;
//    printf("Hello");
```

```
    a = 50;
    b = 30;
    c = a + b;
    if(a > c){
        printf("Got it!");
    }
    return 0;
}
```

Output:



```
CDLab — -zsh — 80×24
vaporcrash@Sharvans-MacBook-Pro CDLab % ./a
PDIR
PDIR
KW  ID SP SP
    KW  ID  ID  ID
SNGLINE

    ID  ASSIGN  NUMCONST NUMCONST SP
    ID  ASSIGN  NUMCONST NUMCONST SP
    ID  ASSIGN  ID  ARITHOP  ID
    KW ID  RELOP  ID SP
        FC SP
    SP
    KW  NUMCONST SP
SP
vaporcrash@Sharvans-MacBook-Pro CDLab %
```

Learning objective:
Learn to parse and identify tokens in a given program, and match regular expressions to build a working lexical analyser.