CD Lab Programs

WEEK-1: Design a Lexical analyzer for C language. The lexical analyzer should ignore redundant spaces, tabs and newlines. It should also ignore comments. Although the syntax specification states that identifiers can be arbitrarily long, you may restrict the length to some reasonable value.

(a) Is constant or not

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
   char s[50];
   printf("enter string:");
   gets(s);
   if(atoi(s))
      printf("given string is contant");
   else
      printf("given string is not constant");
}
```

```
enter string:123
given string is constant
```

```
enter string:abc
given string is not constant
```

(b)Checking comment lines

```
#include<stdio.h>
//#include<stdlib.h>
#include<string.h>

void main()
{
   char s[50];
```

```
    printf("enter input:");
    gets(s);
    if(s[0]=='/')
    {
       if(s[1]=='/')
          printf("given statement is a contant");
       else if(s[1]=='*')
          {
             int flag=0,n=strlen(s)-1;
             if(s[n]=='/' && s[n-1]=='*')
                printf("given statement is a comment");
             else{
                printf("given statement is not a comment");
             }
          }
          else
             printf("given statement is not a comment");
    }
    else
    printf("given statement is not a comment");
}
```

(c)Checking identifies

```
#include<stdio.h>
//#include<stdlib.h>
#include<string.h>
```

```
void main()
{
   char s[50];
   printf("enter input:");
   gets(s);
   int flag=0;
   if(isalpha(s[0])||s[0]=='_')
      {
         for(int i=0;i<strlen(s);i++){
            if(isdigit(s[i])|| isalpha(s[i])|| s[i]=='_')
               flag=1;
            else
               break;
         }
      }
      if(flag==1)
         printf("valid");
      else
         printf("invalid");
}
```

(d)checking keywords

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
```

```
void main()
{
   char
kw[21][10]={"auto","double","struct","break","else","long","switch","case","enum","register","type
def","char","extern","return","union","const","float","short","do","if","while"};
   char s[100],tokens[25][25];
   int j=0,k=0,wc=0,flag=0;
   printf("enter the c statement:");
   gets(s);
   for(int i=0;i<strlen(s);i++){
      if(s[i]!=' '){
         tokens[j][k]=s[i];
         k++;
      }
      if(s[i]==' ')
      {
         tokens[j][k]='\0';
         j++;
         k=0;
         wc++;
         }
   }
   tokens[j][k]='\0';
   for(int i=0;i<wc;i++){
      for(int j=0;j<32;j++){
         if(strcmp(kw[j],tokens[i])==0){
            printf("%s is a keyword",tokens[i]);
            flag=1;
         }
      }
   }
   if(flag==0)
```

```
   printf("there are no keywords");
}
```

(e)Checking operators

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
char s[5];
//clrscr();
printf("\n Enter any operator:");
gets(s);
switch(s[0])
{
case'>':
if(s[1]=='=')
printf("\n Greater than or equal");
else printf("\n Greater than");
break;
case'<':
if(s[1]=='=')
printf("\n Less than or equal");
else printf("\nLess than");
break;
case'=':
if(s[1]=='=')
printf("\nEqual to");
else
printf("\nAssignment");
break;
case'!':
if(s[1]=='=')
```

```
printf("\nNot Equal");
else
printf("\n Bit Not");
break;
case'&':
if(s[1]=='&')
printf("\nLogical AND");
else printf("\n Bitwise AND");
break;
case'|':
if(s[1]=='|')
printf("\nLogical OR");
else
printf("\nBitwise OR");
break;
case'+':
printf("\n Addition");
break;
case'-':
printf("\nSubstraction");
break;
case'*':
printf("\nMultiplication");
break;
case'/':
printf("\nDivision");
break;
case'%':
printf("Modulus");
break;
default:
```

```
printf("\n Not a operator");
}
//getch();
}
```