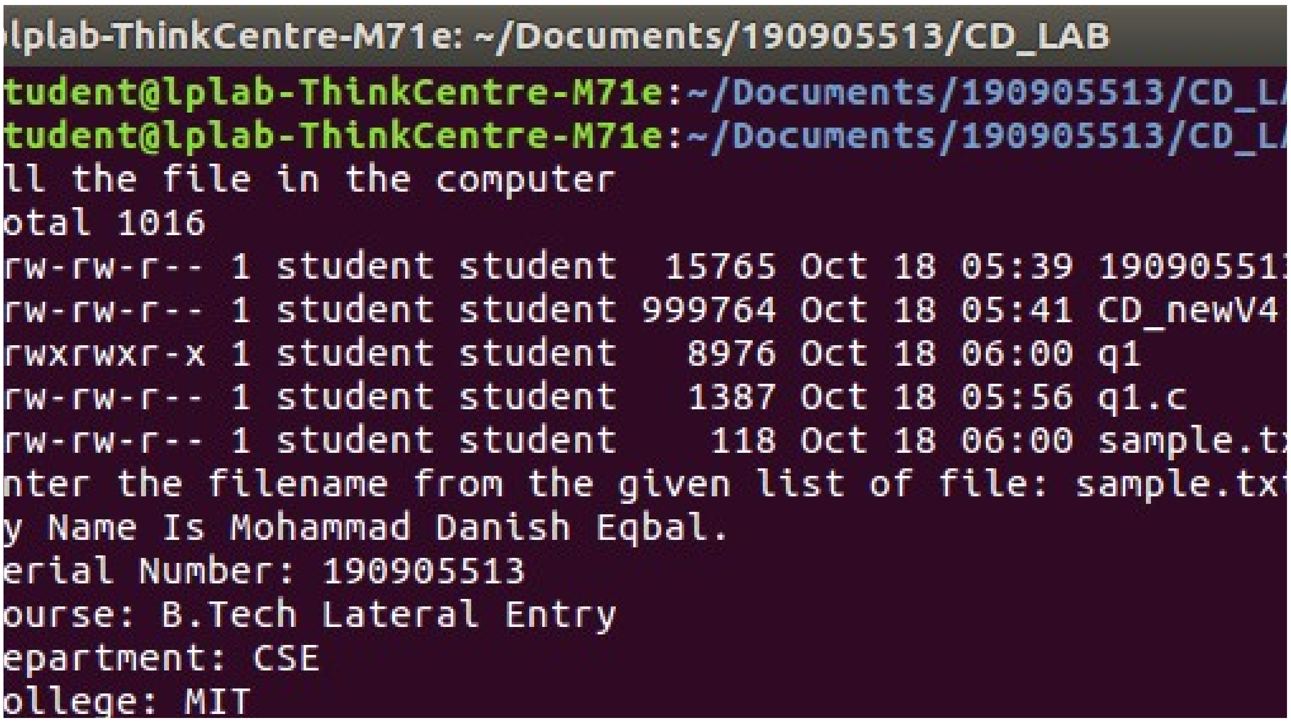
## SAMPLE EXERCISE:

**Write a program in ‘C’ to identify the arithmetic and relational operators from the given input ‘C’ file.**

**Program:**

```
//Program in ‘C’ to identify the arithmetic and relational
operators from the given input ‘C’ file.
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
int main()
{
char c,buf[10];
FILE *fp=fopen("digit.c","r");
c = fgetc(fp);
if (fp == NULL)
{
printf("Cannot open file \n");
exit(0);
}
while(c!=EOF)
{
int i=0;
buf[0]='\0';
if(c=='=')
{
buf[i++]=c;
c = fgetc(fp);
if(c=='=')
{
buf[i++]=c;
buf[i]='\0';
printf("\n Relational operator : %s",buf);
}
else
{
buf[i]='\0';
printf("\n Assignment operator: %s",buf);
}
}
else
{
if(c=='<'||c=='>'||c=='!')
{
buf[i++]=c;
c = fgetc(fp);
if(c=='=')
{
buf[i++]=c;
}
buf[i]='\0';
printf("\n Relational operator : %s",buf);
}
else
{
buf[i]='\0';
}
}
c = fgetc(fp);
} }
```

**Output:**



```
lplab-ThinkCentre-M71e: ~/Documents/190905513/CD_LAB
tudent@lplab-ThinkCentre-M71e:~/Documents/190905513/CD_L
tudent@lplab-ThinkCentre-M71e:~/Documents/190905513/CD_L
ll the file in the computer
otal 1016
rw-rw-r-- 1 student student  15765 Oct 18 05:39 19090551
rw-rw-r-- 1 student student 999764 Oct 18 05:41 CD_newV4
rwxrwxr-x 1 student student   8976 Oct 18 06:00 q1
rw-rw-r-- 1 student student   1387 Oct 18 05:56 q1.c
rw-rw-r-- 1 student student    118 Oct 18 06:00 sample.t
nter the filename from the given list of file: sample.tx
y Name Is Mohammad Danish Eqbal.
erial Number: 190905513
ourse: B.Tech Lateral Entry
epartment: CSE
ollege: MIT
```

**LAB EXERCISES:**

**1. Design a lexical analyzer which contains getNextToken( ) for a simple C program to create a structure of token each time and return, which includes row number, column number and token type. The tokens to be identified are arithmetic operators, relational operators, logical operators, special symbols, keywords, numerical constants, string literals and identifiers. Also, getNextToken() should ignore all the tokens when encountered inside single line or multiline comment or inside string literal. Preprocessor directive should also be stripped.**

**Program:**

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

#define FILEINPUT "digit.c"

struct token
{
    char lexeme[64];
    int row,col;
    char type[20];
};

static int row=1,col=1;
char buf[2048];

const char specialsymbols[]={'?',';',':',','};
const char *keywords[] = {"const", "char", "int","return","for",
"while", "do",
                                   "switch", "if", "else","unsigned",
"case", "break" };

const char arithmeticsymbols[]={'*'};

int isKeyword(const char *str)
{
    for(int i=0;i<sizeof(keywords)/sizeof(char*);i++)
    {
    if(strcmp(str,keywords[i])==0)
      {
      return 1;
      }
    }
    return 0;
}

int charBelongsTo(int c,const char *arr)
{
    int len;

  if(arr==specialsymbols)
  {
      len=sizeof(specialsymbols)/sizeof(char);
    }

  else if(arr==arithmeticsymbols)
```

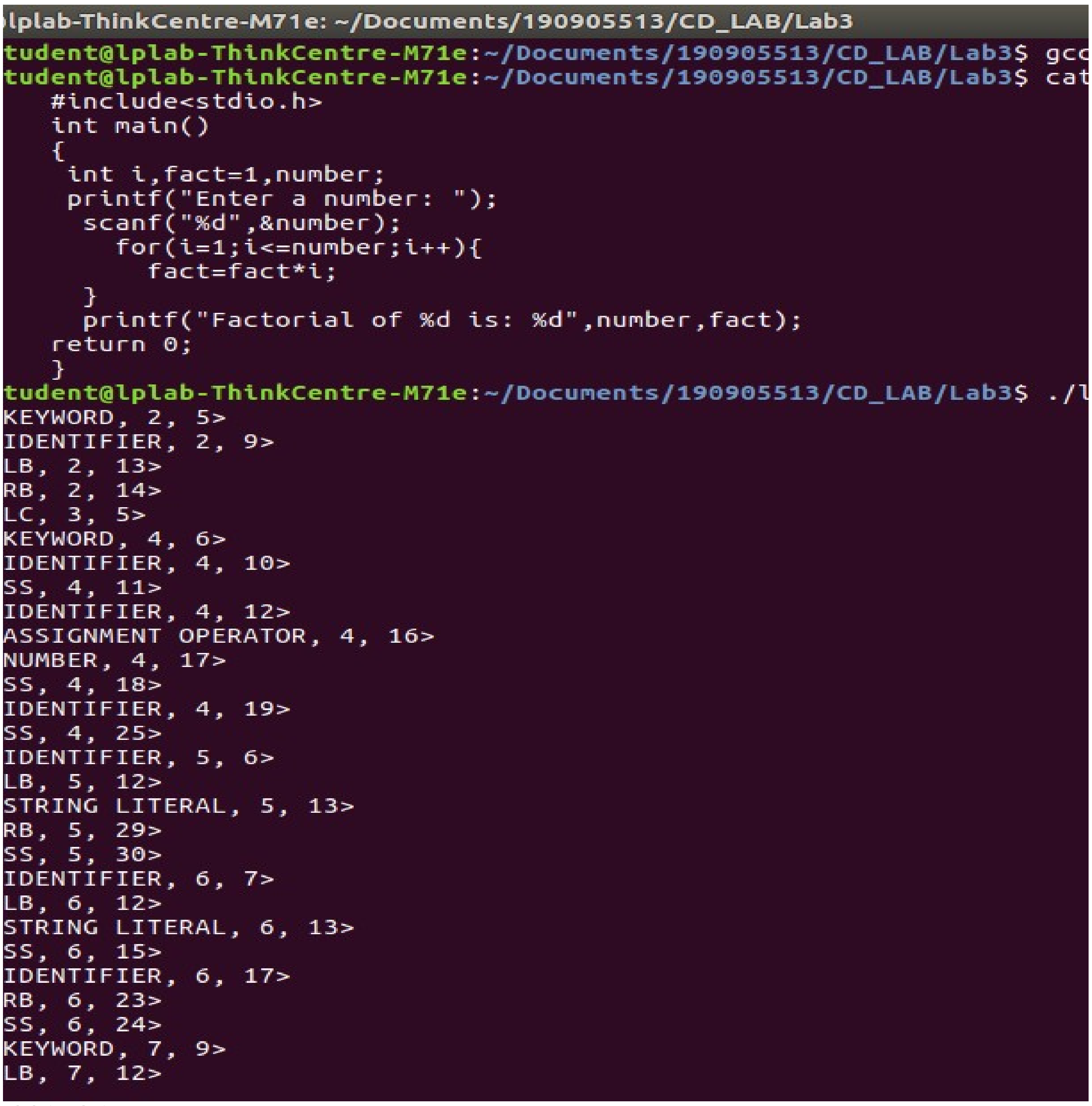
```
    {
      len=sizeof(arithmeticsymbols)/sizeof(char);
    }

  for(int i=0;i<len;i++)
    {
      if(c==arr[i])
      {
        return 1;
      }
    }

  return 0;
}

void fillToken(struct token *tkn,char c,int row,int col, char
*type)
{
    tkn->row=row;
    tkn->col=col;
    strcpy(tkn->type,type);
    tkn->lexeme[0]=c;
    tkn->lexeme[1]='\0';
}

void newLine()
{
    ++row;
    col=1;
}

struct token getNextToken(FILE *f1)
{
    int c;
    struct token tkn=
    {
        .row=-1
    };

  int gotToken=0;

  while(!gotToken && (c=fgetc(f1))!=EOF)
  {
        if(charBelongsTo(c,specialsymbols))
        {
             fillToken(&tkn,c,row,col,"SS");
             gotToken=1;
             ++col;
        }

    else if(charBelongsTo(c,arithmeticsymbols))
        {
          fillToken(&tkn,c,row,col,"ARITHMETIC OPERATOR");
          gotToken=1;
             ++col;
        }

    else if(c=='(')
        {
          fillToken(&tkn,c,row,col,"LB");
```

```c
          gotToken=1;
       ++col;
     }

 else if(c==')')
   {
       fillToken(&tkn,c,row,col,"RB");
          gotToken=1;
       ++col;
 }

 else if(c=='{')
     {
       fillToken(&tkn,c,row,col,"LC");
          gotToken=1;
          ++col;
     }

 else if(c=='}')
     {
       fillToken(&tkn,c,row,col,"RC");
          gotToken=1;
          ++col;
     }

 else if(c=='+')
     {
          int d=fgetc(f1);

   if(d!='+')
       {
              fillToken(&tkn,c,row,col,"ARITHMETIC OPERATOR");
              gotToken=1;
              ++col;
            fseek(f1,-1,SEEK_CUR);
          }

   else
       {
            fillToken(&tkn,c,row,col,"UNARY OPERATOR");
            strcpy(tkn.lexeme,"++");
            gotToken=1;
            col+=2;
          }
      }

  else if(c=='-')
      {
           int d=fgetc(f1);

    if(d!='-')
           {
               fillToken(&tkn,c,row,col,"ARITHMETIC OPERATOR");
               gotToken=1;
               ++col;
               fseek(f1,-1,SEEK_CUR);
             }

     else
             {
                    fillToken(&tkn,c,row,col,"UNARY OPERATOR");
                    strcpy(tkn.lexeme,"--");
                  gotToken=1;
                    col+=2;
                    }
               }

      else if(c=='=')
               {
                   int d=fgetc(f1);

        if(d!='=')
                   {
                     fillToken(&tkn,c,row,col,"ASSIGNMENT
OPERATOR");

                     gotToken=1;
                     ++col;
                     fseek(f1,-1,SEEK_CUR);
                   }

        else
                   {
                    fillToken(&tkn,c,row,col,"RELATIONAL OPERATOR");
                    strcpy(tkn.lexeme,"==");
                    gotToken=1;
                    col+=2;
                   }
        }

     else if(isdigit(c))
         {
             tkn.row=row;
             tkn.col=col++;
             tkn.lexeme[0]=c;

       int k=1;

       while((c=fgetc(f1))!=EOF && isdigit(c))
             {
               tkn.lexeme[k++]=c;
               col++;
             }

       tkn.lexeme[k]='\0';
             strcpy(tkn.type,"NUMBER");
             gotToken=1;
             fseek(f1,-1,SEEK_CUR);
     }

     else if(c == '#')
       {
             while((c = fgetc(f1)) != EOF && c != '\n');
             newLine();
         }

     else if(c=='\n')
         {
             newLine();
             c = fgetc(f1);
```

```
    if(c == '#')
          {
             while((c = fgetc(f1)) != EOF && c != '\n');
                newLine();
          }

    else if(c != EOF)
          {
             fseek(f1, -1, SEEK_CUR);
          }
      }

 else if(isspace(c))
   {
         ++col;
     }

 else if(isalpha(c)||c=='_')
   {
         tkn.row=row;
         tkn.col=col++;
         tkn.lexeme[0]=c;
         int k=1;

   while((c=fgetc(f1))!= EOF && isalnum(c))
         {
           tkn.lexeme[k++]=c;
           ++col;
         }

   tkn.lexeme[k]='\0';

   if(isKeyword(tkn.lexeme))
         {
           strcpy(tkn.type,"KEYWORD");
         }

   else
         {
           strcpy(tkn.type,"IDENTIFIER");
         }

   gotToken=1;
         fseek(f1,-1,SEEK_CUR);
      }

 else if(c=='/')
   {
      int d=fgetc(f1);
      ++col;//Do we check EOF here?

  if(d=='/')
      {
          while((c=fgetc(f1))!= EOF && c!='\n')
          {
            ++col;
          }

    if(c=='\n')
          {
                 newLine();
               }
             }

        else if(d=='*')
             {
               do
               {
                 if(d=='\n')
                 {
                      newLine();
                 }

           while((c==fgetc(f1))!= EOF && c!='*')
                 {
                        ++col;

             if(c=='\n')
                   {
                          newLine();
                      }
                    }

           ++col;
               }while((d==fgetc(f1))!= EOF && d!='/' && (++col));
               ++col;
              }

        else
             {
               fillToken(&tkn,c,row,--col,"ARITHMETIC OPERATOR");
               gotToken=1;
               fseek(f1,-1,SEEK_CUR);
             }
         }

     else if(c == '"')
       {
                 tkn.row = row;
                 tkn.col = col;
                 strcpy(tkn.type, "STRING LITERAL");
                 int k = 1;
                 tkn.lexeme[0] = '"';

         while((c = fgetc(f1)) != EOF && c != '"')
                 {
                     tkn.lexeme[k++] = c;
                     ++col;
                 }

         tkn.lexeme[k] = '"';
                 gotToken = 1;
         }

      else if(c == '<' || c == '>' || c == '!')
           {
                 fillToken(&tkn, c, row, col, "RELATIONAL
OPERATOR");
                 ++col;
                 int d = fgetc(f1);
```

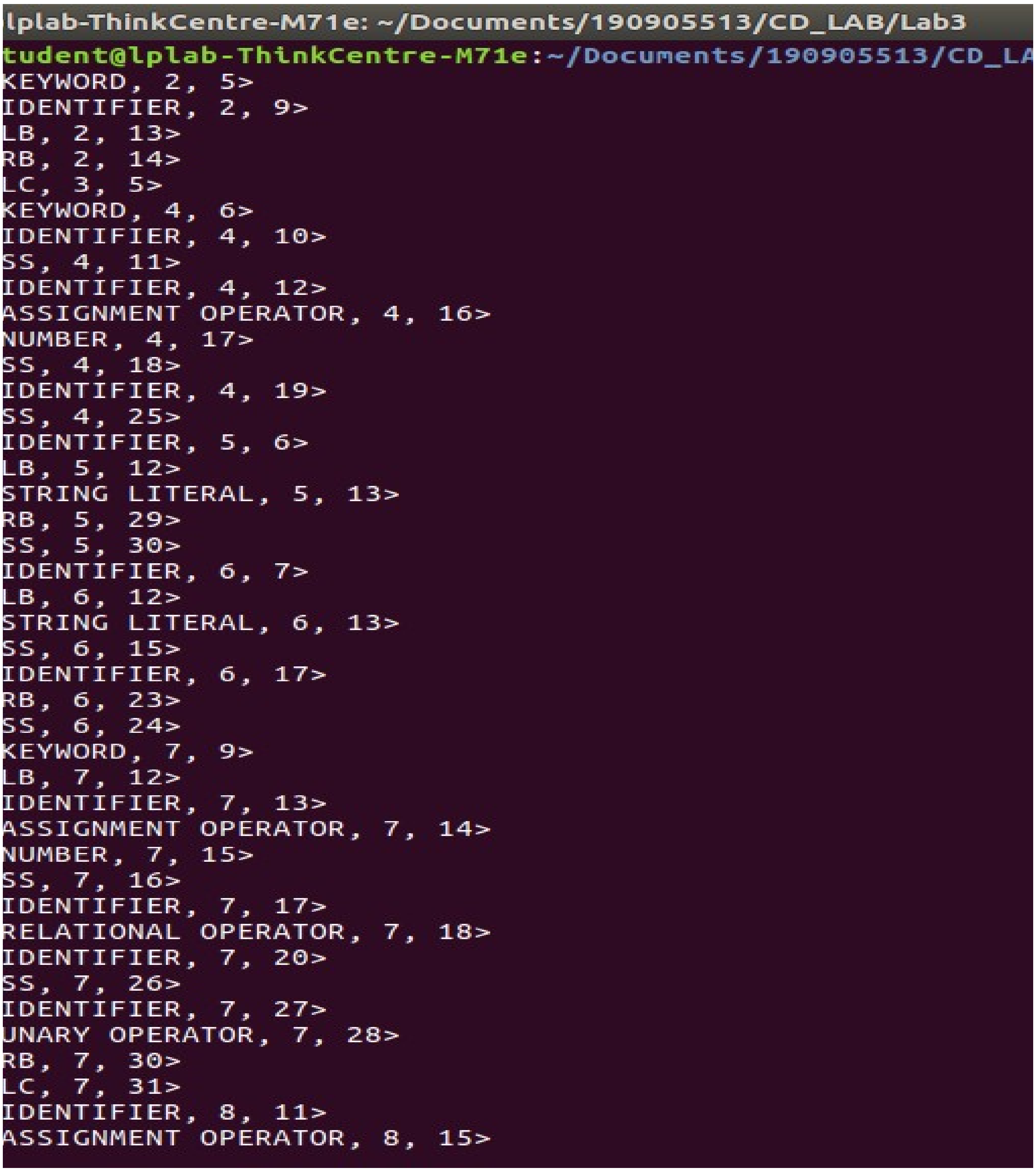
```
                if(d == '=')
                {
                    ++col;
                    strcat(tkn.lexeme, "=");
                }

        else
                {
                    if(c == '!')
                    {
                        strcpy(tkn.type, "LOGICAL OPERATOR");
                    }

          fseek(f1, -1, SEEK_CUR);
                 }

         gotToken = 1;
            }

    else if(c == '&' || c == '|')
        {
                int d = fgetc(f1);

        if(c == d)
                {
                    tkn.lexeme[0] = tkn.lexeme[1] = c;
                    tkn.lexeme[2] = '\0';
                    tkn.row = row;
                    tkn.col = col;
                    ++col;
                    gotToken = 1;
                    strcpy(tkn.type, "LOGICAL OPERATOR");
                }

        else
                {
                    fseek(f1, -1, SEEK_CUR);
                }
                ++col;
        }

    else
        {
            ++col;
        }
    }
    return tkn;
}

int main()
{
    FILE *f1=fopen(FILEINPUT,"r");
    if(f1==NULL)
    {
      printf("Error! File cannot be opened!\n");
      return 0;
    }

  struct token tkn;
    while((tkn=getNextToken(f1)).row!=-1)
    {
        printf("<%s, %d, %d>\n",tkn.type, tkn.row,tkn.col);
  }

  fclose(f1);
}
```

**Output:**



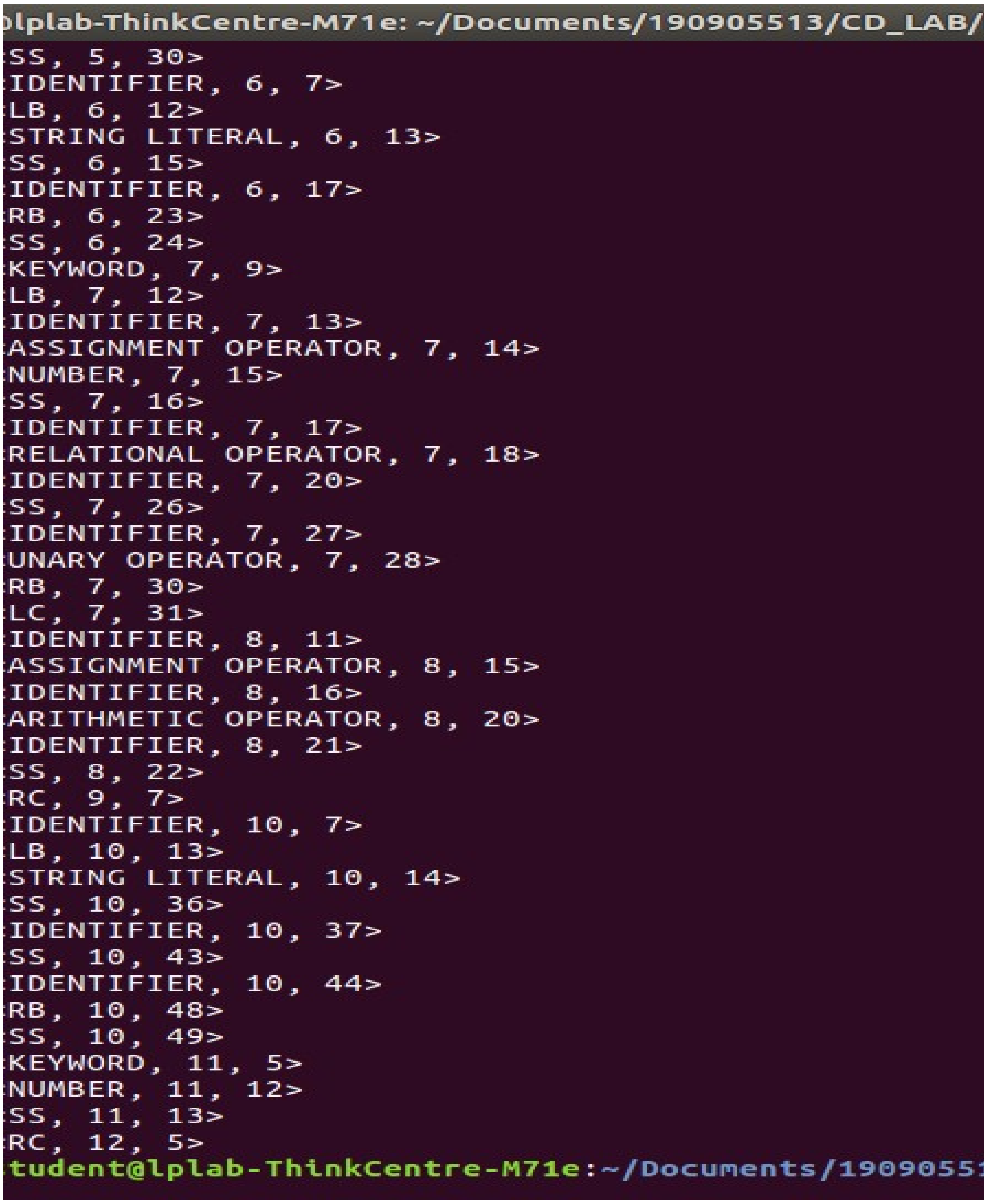
```
lplab-ThinkCentre-M71e: ~/Documents/190905513/CD_LAB/Lab3
tudent@lplab-ThinkCentre-M71e:~/Documents/190905513/CD_LAB/Lab3$ gcc
tudent@lplab-ThinkCentre-M71e:~/Documents/190905513/CD_LAB/Lab3$ cat
   #include<stdio.h>
   int main()
   {
    int i,fact=1,number;
    printf("Enter a number: ");
     scanf("%d",&number);
       for(i=1;i<=number;i++){
         fact=fact*i;
     }
     printf("Factorial of %d is: %d",number,fact);
   return 0;
   }
tudent@lplab-ThinkCentre-M71e:~/Documents/190905513/CD_LAB/Lab3$ ./l
KEYWORD, 2, 5>
IDENTIFIER, 2, 9>
LB, 2, 13>
RB, 2, 14>
LC, 3, 5>
KEYWORD, 4, 6>
IDENTIFIER, 4, 10>
SS, 4, 11>
IDENTIFIER, 4, 12>
ASSIGNMENT OPERATOR, 4, 16>
NUMBER, 4, 17>
SS, 4, 18>
IDENTIFIER, 4, 19>
SS, 4, 25>
IDENTIFIER, 5, 6>
LB, 5, 12>
STRING LITERAL, 5, 13>
RB, 5, 29>
SS, 5, 30>
IDENTIFIER, 6, 7>
LB, 6, 12>
STRING LITERAL, 6, 13>
SS, 6, 15>
IDENTIFIER, 6, 17>
RB, 6, 23>
SS, 6, 24>
KEYWORD, 7, 9>
LB, 7, 12>
```

//(1/3)



```
lplab-ThinkCentre-M71e: ~/Documents/190905513/CD_LAB/Lab3
tudent@lplab-ThinkCentre-M71e:~/Documents/190905513/CD_LA
KEYWORD, 2, 5>
IDENTIFIER, 2, 9>
LB, 2, 13>
RB, 2, 14>
LC, 3, 5>
KEYWORD, 4, 6>
IDENTIFIER, 4, 10>
SS, 4, 11>
IDENTIFIER, 4, 12>
ASSIGNMENT OPERATOR, 4, 16>
NUMBER, 4, 17>
SS, 4, 18>
IDENTIFIER, 4, 19>
SS, 4, 25>
IDENTIFIER, 5, 6>
LB, 5, 12>
STRING LITERAL, 5, 13>
RB, 5, 29>
SS, 5, 30>
IDENTIFIER, 6, 7>
LB, 6, 12>
STRING LITERAL, 6, 13>
SS, 6, 15>
IDENTIFIER, 6, 17>
RB, 6, 23>
SS, 6, 24>
KEYWORD, 7, 9>
LB, 7, 12>
IDENTIFIER, 7, 13>
ASSIGNMENT OPERATOR, 7, 14>
NUMBER, 7, 15>
SS, 7, 16>
IDENTIFIER, 7, 17>
RELATIONAL OPERATOR, 7, 18>
IDENTIFIER, 7, 20>
SS, 7, 26>
IDENTIFIER, 7, 27>
UNARY OPERATOR, 7, 28>
RB, 7, 30>
LC, 7, 31>
IDENTIFIER, 8, 11>
ASSIGNMENT OPERATOR, 8, 15>
```

//(2/3)



```
:lplab-ThinkCentre-M71e: ~/Documents/190905513/CD_LAB/
:SS, 5, 30>
:IDENTIFIER, 6, 7>
:LB, 6, 12>
:STRING LITERAL, 6, 13>
:SS, 6, 15>
:IDENTIFIER, 6, 17>
:RB, 6, 23>
:SS, 6, 24>
:KEYWORD, 7, 9>
:LB, 7, 12>
:IDENTIFIER, 7, 13>
:ASSIGNMENT OPERATOR, 7, 14>
:NUMBER, 7, 15>
:SS, 7, 16>
:IDENTIFIER, 7, 17>
:RELATIONAL OPERATOR, 7, 18>
:IDENTIFIER, 7, 20>
:SS, 7, 26>
:IDENTIFIER, 7, 27>
:UNARY OPERATOR, 7, 28>
:RB, 7, 30>
:LC, 7, 31>
:IDENTIFIER, 8, 11>
:ASSIGNMENT OPERATOR, 8, 15>
:IDENTIFIER, 8, 16>
:ARITHMETIC OPERATOR, 8, 20>
:IDENTIFIER, 8, 21>
:SS, 8, 22>
:RC, 9, 7>
:IDENTIFIER, 10, 7>
:LB, 10, 13>
:STRING LITERAL, 10, 14>
:SS, 10, 36>
:IDENTIFIER, 10, 37>
:SS, 10, 43>
:IDENTIFIER, 10, 44>
:RB, 10, 48>
:SS, 10, 49>
:KEYWORD, 11, 5>
:NUMBER, 11, 12>
:SS, 11, 13>
:RC, 12, 5>
:tudent@lplab-ThinkCentre-M71e:~/Documents/1909055
```

//(3/3)