SYSTEM SOFTWARE AND OPERATING SYSTEM LABORATORY
(15CSL67)

1a. Write a LEX program to recognize valid arithmetic expression. Identifiers in the expression could be only integers and operators could be + and *. Count the identifiers & operators present and print them separately.

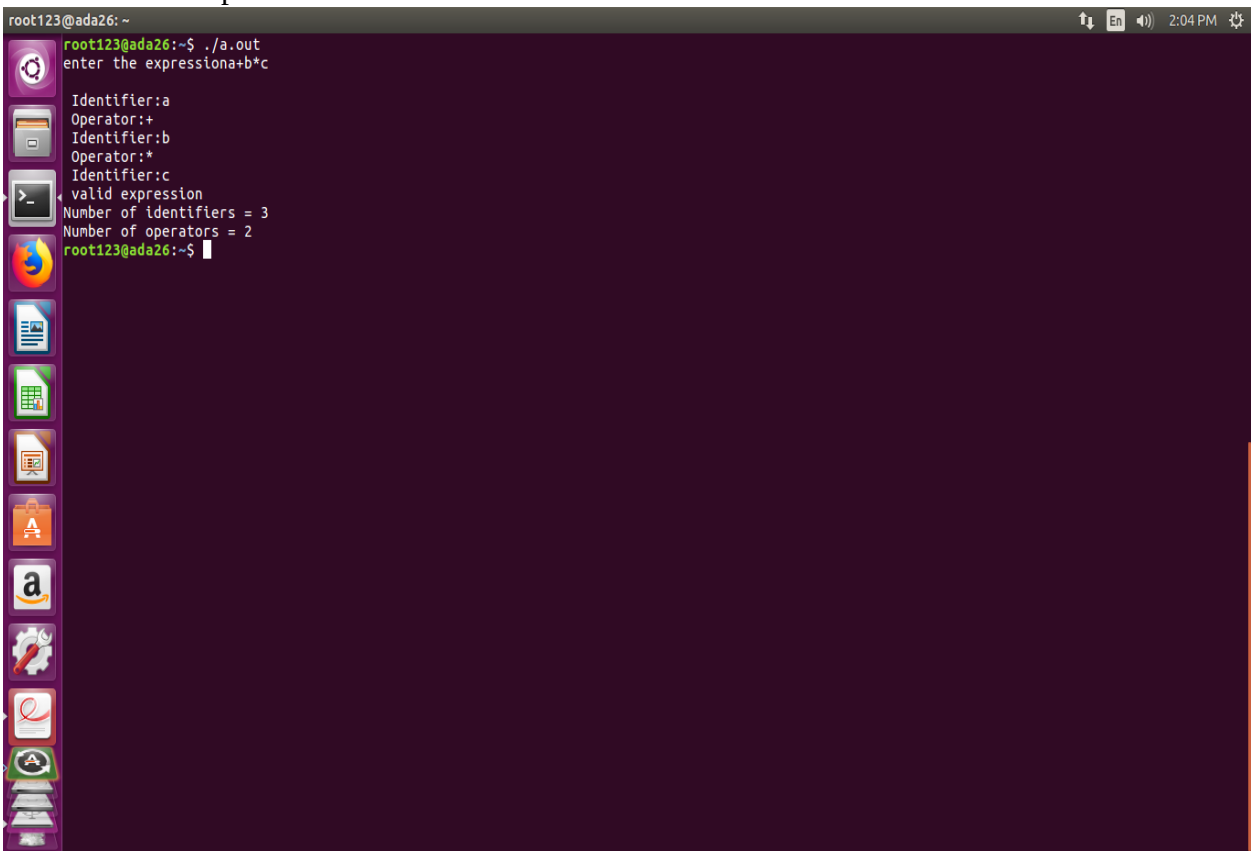
```
%{
#include<stdio.h>
int v=0,op=0,id=0,flag=0;
%}

%%
[a-z A-Z]+[0-9 A-Z a-z]*  {id++;printf("\n Identifier:");ECHO;}
[\+\-\*\/\=]    {op++;printf("\n Operator:");ECHO;}
"("    {v++;}
")"    {v--;}
";"    {flag=1;}
.|\n   {;}
%%

int main()
{
printf("enter the expression");
yylex();

if(((op+1)==id)&&(v==0)&&(flag==0))
{
  printf("\n valid expression\n");
 printf("Number of identifiers = %d\n",id);
 printf("Number of operators = %d\n",op);
}
else
  printf("\n Invalid expression\n");

}
int yywrap()
{
return 1;
}
```

Execution Steps:

Lex <lexfilename.l>

cc lex.yy.c –ll

. /a.out <temp.txt>



b. Write YACC program to evaluate arithmetic expression involving operators: +, -, * and /

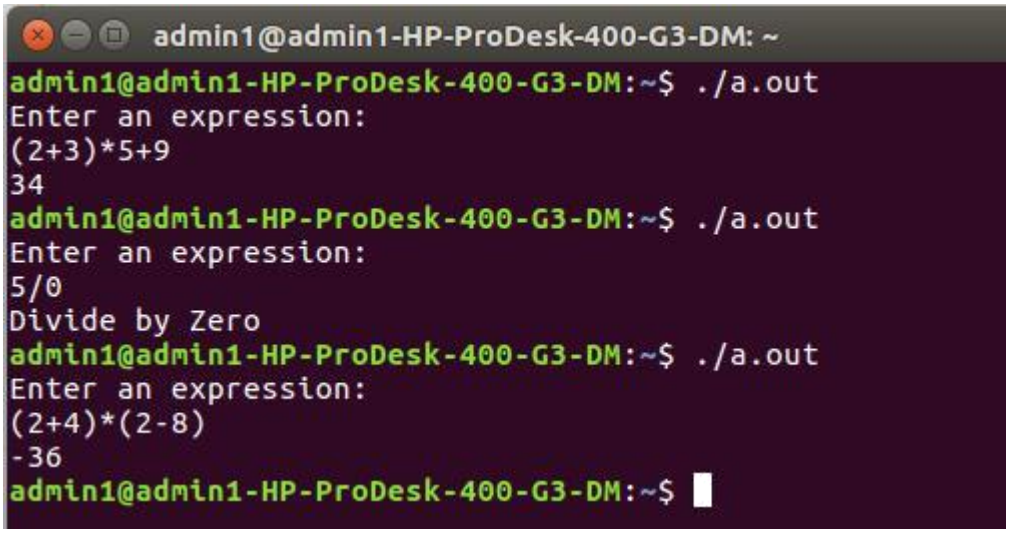
Lex Part

```
%{
#include "y.tab.h"
extern yylval;
%}
%%
[0-9]+ {yylval=atoi(yytext);return num;}  /* convert the
                                            string to number and
                                            send the value*/
[\+\-\*\/] {return yytext[0];}
← {return yytext[0];}
← {return yytext[0];}
. {;}
\n {return 0;}
%%
```

YACC Part

```
%{
#include<stdio.h>
#include<stdlib.h>
%}
%token num
%left '+' '-'
%left '*' '/'
%%
input:exp {printf("%d\n",$$);exit(0);}
exp:exp'+'exp {$$=$1+$3;}
|exp'-'exp{$$=$1-$3;}
|exp'*'exp{$$=$1*$3;}
|exp'/'exp
         { if($3==0){printf("Divide
         by Zero\n");exit(0);} else
          $$=$1/$3;}
|'('exp')'{$$=$2;}
|num{$$=$1;};
%%
int yyerror()
{
printf("error");
exit(0);
}

int main()
```

```
{
printf("Enter an expression:\n");

yyparse();
}
```



2. Develop, Implement and execute a program using YACC tool to recognize all strings ending with *b* preceded by *n a's* using the grammar $a^n b$ (note: input *n* value).

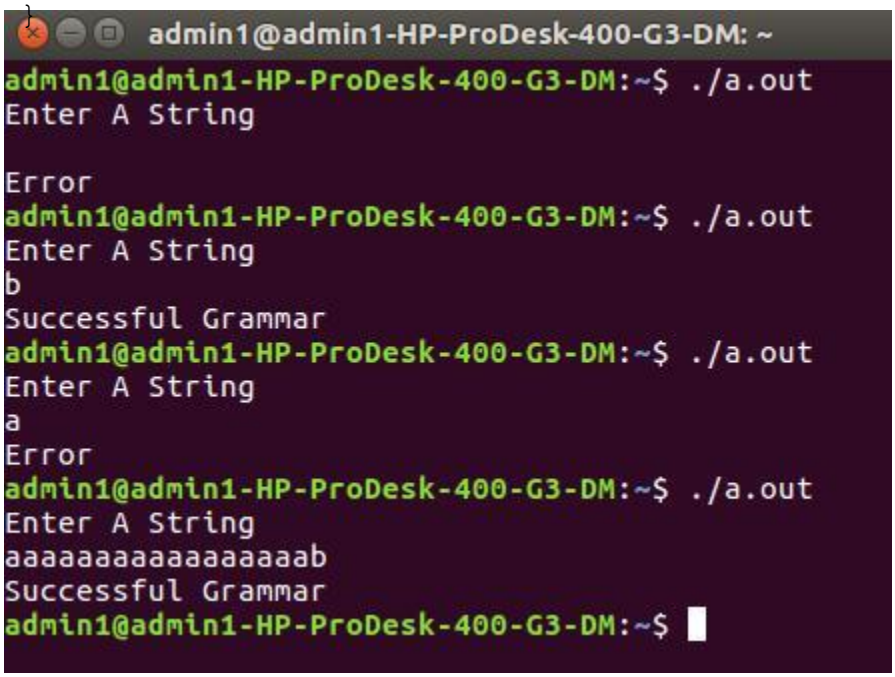
Lex Part

```
%{
#include "y.tab.h"
%}
%%
a {return A;}
b {return B;}
[\n] return '\n';
%%
```

YACC Part

```
%{
#include<stdio.h>
#include<stdlib.h>
%}
%token A B
%%
input:s'\n' {printf("Successful Grammar\n");exit(0);}
s: A

s1
B|
B
s1
:
;
|
A
s1
%%
 main()
{
printf("Enter
A String\n");
yyparse();
}

int yyerror()
{
printf("Error \n");
exit(0);
```

}


admin1@admin1-HP-ProDesk-400-G3-DM: ~
admin1@admin1-HP-ProDesk-400-G3-DM:~$ ./a.out
Enter A String
Error
admin1@admin1-HP-ProDesk-400-G3-DM:~$ ./a.out
Enter A String
b
Successful Grammar
admin1@admin1-HP-ProDesk-400-G3-DM:~$ ./a.out
Enter A String
a
Error
admin1@admin1-HP-ProDesk-400-G3-DM:~$ ./a.out
Enter A String
aaaaaaaaaaaaaaaaab
Successful Grammar
admin1@admin1-HP-ProDesk-400-G3-DM:~$

3. **Design, develop and implement YACC/C program to construct *Predictive / LL(1) Parsing Table* for the grammar rules*: A →aBa , B →bB | ε*. Use this table to parse the sentence*: abba$*.**

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

  char STACK[20]="\0";
  int TOP=-1,flag=0;
  int B_ptr = 0;
  char BUFFER[20],G_prod[20];
  char table [3][3][10] = {
                "NT", "a","b",
                "A", "aBa","Error",
                "B", "Îµ","bB",
                };
  char pop()
  {
   char ch;
   ch = STACK[TOP--];
   return ch;
  }

  void push(char ch)
  {
   STACK[++TOP] = ch;
  }

  void stack_content()
  {
      if (TOP != -1)
      {
              int i = 0;
             printf("\nstack content: ");
              while(i <= TOP)
              {
                  printf("%c",STACK[i++]);
              }
              printf("\n");
      }
      return;
  }

  int isterm(char c)
  {
```

```
     if (c >= 'a' && c <= 'z')
          return 1;
     else
          return 0;
    }

    int Parser_table(char stack_top,char buf_value,int flag)
    {
     int r,c;
      switch(stack_top)
     {
          case 'A' : r = 1; break;
          case 'B' : if(flag<=5) r = 2; else r = 3;
     }
     switch(buf_value)
     {
          case 'a' : c = 1; break;
          case 'b' : c = 2;
     }

          if (strcmp(table[r][c],"error") == 0)
          return 0;

     if (strcmp(table[r][c],"Îµ") != 0)
     {
          strcpy(G_prod,table[r][c]);
     }
     return 1;
    }

    int main()
    {
            int i,j,stln;
            printf("LL(1)  PARSER  TABLE \n");
          for(i=0;i<3;i++)
              {
                for(j=0;j<3;j++)
                    {
                         printf("%s\t",table[i][j]);
                    }
                    printf("\n");
          }
printf("\n");
printf("ENTER THE STRING into the Buffer and also give a ';' as
the terminator: ");
scanf("%s",BUFFER);
printf("\n THE STRING in the Buffer is %s",BUFFER);
```

```
          if(BUFFER[strlen(BUFFER)-1] != ';')
          {
              printf("END OF STRING MARKER SHOULD BE ';'");
              exit(0);
          }
          push('$');
          push('A');
          while(STACK[TOP] != '$') // Stack is not Empty
         {
             flag++;
               if (STACK[TOP] == BUFFER[B_ptr]) // X is a
               {
                    printf("\n1.The poped item is - %c,",pop());
                    B_ptr++;
printf("\t buffer cont - %.*s",strlen(BUFFER),BUFFER+B_ptr);
               }
               else if(isterm(STACK[TOP])) // is X is terminal
                    {
                        printf("\n2. $ %c",STACK[TOP]);
                        printf("\t Error in Parsing \n");
                    }
               else
if (!Parser_table(STACK[TOP],BUFFER[B_ptr],flag))
printf("3. Error Entry in Parse Table ");
else
if (Parser_table(STACK[TOP],BUFFER[B_ptr],flag))
          {
          if (flag < 6 && strcmp(G_prod,"Îµ") != 0)
          {
          printf("\n4.1 flag = %d, prod id- %s*\t",flag,G_prod);
           pop();
           stln = strlen(G_prod);
           for(i=stln-1;i>=0;i--)
           push(G_prod[i]);
           stack_content();
           }
          else
          {
        stack_content();
        printf("\n4.2 flag = %d  *reduce by %s*",flag,"B->Îµ");
        pop();
        printf("\t buffer content is %c",BUFFER[B_ptr]);
                        }
                    }
               }
          if (STACK[TOP] == '$' && BUFFER[B_ptr] == ';')
```

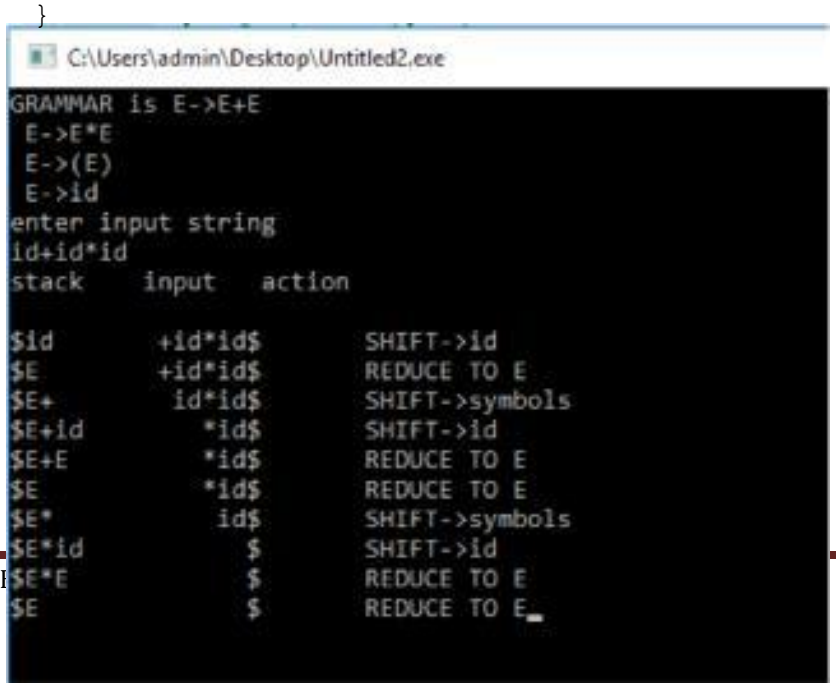
```
    printf("\n** The string is accepted **");
    else
    printf("\n** The string is not accepted **");

   }
```

```
LL(1)  PARSER  TABLE
NT      a       b
A       aBa     Error
B       ╬       bB

ENTER THE STRING into the Buffer and also give a ';' as the terminator: abba;

 THE STRING in the Buffer is abba;
4.1 flag = 1, prod id- aBa*
stack content: $aBa

1.The poped item is - a,         buffer cont - bba;
4.1 flag = 3, prod id- bB*
stack content: $aBb

1.The poped item is - b,         buffer cont - ba;
4.1 flag = 5, prod id- bB*
stack content: $aBb

1.The poped item is - b,         buffer cont - a;
stack content: $aB

4.2 flag = 7  *reduce by B->╬*  buffer content is a
1.The poped item is - a,         buffer cont - ;
** The string is accepted **
Process returned 29 (0x1D)   execution time : 3.459 s
Press any key to continue.
```

4. Design, develop and implement YACC/C program to demonstrate ***Shift Reduce Parsing*** technique for the grammar rules: ***E →E+T | T, T →T*F | F, F → (E) | id*** and parse the sentence***: id + id * id***.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
int k=0,z=0,i=0,j=0,c=0;
char a[16],ac[20],stk[15],act[10];
void check();
void main()
  {
   puts("GRAMMAR is E->E+E \n E->E*E \n E->(E) \n E->id");
   puts("enter input string ");
   gets(a);
   c=strlen(a);
   strcpy(act,"SHIFT->");
   puts("stack \t input \t action");
   for(k=0,i=0; j<c; k++,i++,j++)
    {
     if(a[j]=='i' && a[j+1]=='d')
      {
       stk[i]=a[j];
       stk[i+1]=a[j+1];
       stk[i+2]='\0';
       a[j]=' ';
       a[j+1]=' ';
       printf("\n$%s\t%s$\t%sid",stk,a,act);
       check();
    }
else
        {
       stk[i]=a[j];
       stk[i+1]='\0';
       a[j]=' ';
       printf("\n$%s\t%s$\t%ssymbols",stk,a,act);
       check();
      }
    }
    getch();
  }
 void check()
  {
   strcpy(ac,"REDUCE TO E");
   for(z=0; z<c; z++)
    if(stk[z]=='i' && stk[z+1]=='d')
```

```
      {
       stk[z]='E';
       stk[z+1]='\0';
       printf("\n$%s\t%s$\t%s",stk,a,ac);
       j++;
      }
    for(z=0; z<c; z++)
     if(stk[z]=='E' && stk[z+1]=='+' && stk[z+2]=='E')
      {
       stk[z]='E';
       stk[z+1]='\0';
       stk[z+2]='\0';
       printf("\n$%s\t%s$\t%s",stk,a,ac);
       i=i-2;
      }
    for(z=0; z<c; z++)
     if(stk[z]=='E' && stk[z+1]=='*' && stk[z+2]=='E')
      {
       stk[z]='E';
       stk[z+1]='\0';
       stk[z+1]='\0';
       printf("\n$%s\t%s$\t%s",stk,a,ac);
       i=i-2;
      }
    for(z=0; z<c; z++)
     if(stk[z]=='(' && stk[z+1]=='E' && stk[z+2]==')')
      {
       stk[z]='E';
       stk[z+1]='\0';
stk[z+1]='\0';
         printf("\n$%s\t%s$\t%s",stk,a,ac);
       i=i-2;
      }
  }
```



5. Design, develop and implement a C/Java program to generate the machine code using ***Triples*** for the statement ***A = -B * (C +D)*** whose intermediate code in three-address form:

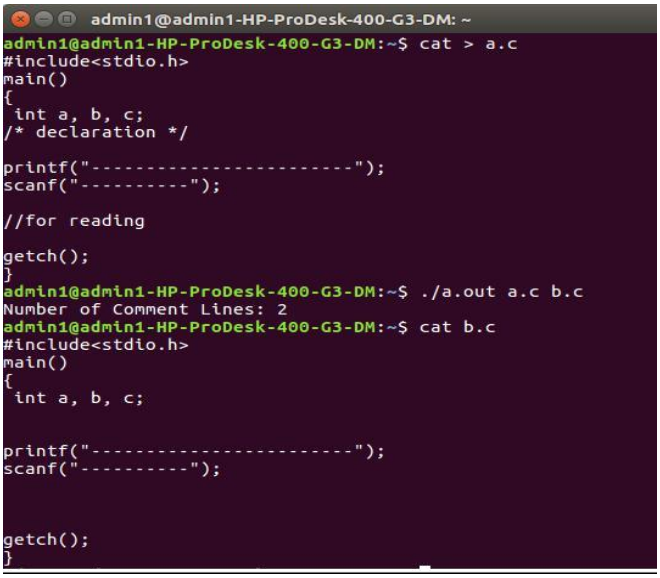
T1 = -B
T2 = C + D
T3 = T1 * T2
A = T3

```
#include<stdio.h>
 #include<stdlib.h>
 #include<ctype.h>
#include<string.h>
char tset[4][3][3]= { {"-","B","?"}, {"+","C","D"},
{"*","0","1"}, {"=","A","2"} };
int main()
{
     int row,col;
      for(row=0;row<4;row++)
     {
     col=2;
     if (tset[row][col][0]=='?')
           {
          printf("\nLD R0,%s%s",tset[row][0],tset[row][1]);
          }
          else
           {
          if(tset[row][0][0]=='+')
               {
               printf("\nLD R1,%s",tset[row][1]);
               printf("\nLD R2,%s",tset[row][2]);
               printf("\nADD R1,R1,R2");
               }
          else
                {
               if(tset[row][0][0]=='*')
                {
               printf("\nMUL R1,R1,R0");
               }
                else
                {
                printf("\nST %s,R1",tset[row][1]);
               }
           }
      }
 }
 printf("\n"); return 0;
 }
```

```
LD R0,-B
LD R1,C
LD R2,D
ADD R1,R1,R2
MUL R1,R1,R0
ST  A,R0
```

6. a) Write a LEX program to eliminate *comment lines* in a *C* program and copy the resulting program into a separate file.

```
%{
#include<stdio.h>
int c_count=0;
%}

%%
"/*"[^*/]*"*/" {c_count++;}
"//".*          {c_count++;}
%%
int main( int argc, char **argv)
{
 FILE *f1,*f2;
if(argc>1)
{
  f1=fopen(argv[1],"r");     /*open first file for reading*/
  if(!f1)                   /*not able to open file*/
{
   printf("file error \n");
     exit(1);
}
yyin=f1;
 f2=fopen(argv[2],"w"); /*open second file for writing*/
 if(!f2)                /*not able to open file*/
 {
printf("Error");
  exit(1);
 }
yyout=f2;

yylex();
printf("Number of Comment Lines: %d\n",c_count);
}
return 0;
}
```



b) Write YACC program to recognize valid *identifier, operators* and *keywords* in the given text (C program) file.

Lex File

```
%{
#include <stdio.h>
#include "y.tab.h"
extern yylval;
%}
%%
[ \t] ;
[+|-|*|/|=|<|>] {printf("operator is %s\n",yytext);return OP;}
[0-9]+ {yylval = atoi(yytext); printf("numbers
is %d\n",yylval); return DIGIT;}
int|char|bool|float|void|for|do|while|if|else|return|void
{printf("keyword is
%s\n",yytext);return KEY;}
[a-zA-Z0-9]+ {printf("identifier is %s\n",yytext);return ID;}
. ;
%%
```

Yacc File

```
%{
#include <stdio.h>
#include <stdlib.h>
int id=0, dig=0, key=0, op=0;
%}
%token DIGIT ID KEY OP
%%
```

```
  input:
  DIGIT input { dig++; }
  | ID input { id++; }
  | KEY input { key++; }
  | OP input {op++;}
  | DIGIT { dig++; }
  | ID { id++; }
  | KEY { key++; }
  | OP { op++;}
  ;
  %%
  #include <stdio.h>
  extern int yylex();
  extern int yyparse();
  extern FILE *yyin;

main()
 {
FILE *myfile = fopen("sam_input.c", "r");
  if (!myfile) {
  printf("I can't open sam_input.c!");
  return -1;
  }
  yyin = myfile;
  do {
  yyparse();
  } while (!feof(yyin));
  printf("numbers = %d\nKeywords = %d\nIdentifiers
  = %d\noperators = %d\n",
  dig, key,id, op);
  }
  void yyerror() {
  printf("EEK, parse error! Message: ");
  exit(-1);
  }
```

```
1 void main()
2 {
3     float a123;
4     char a;
5     char b123;
6     char c;
7     if (sum == 10)
8         printf("pass");
9     else
10        printf("fail");
11 }
```

```
admin1@admin1-HP-ProDesk-400-G3-DM:~$ ./a.out
keyword is void
identifier is main


keyword is float
identifier is a123

keyword is char
identifier is a

keyword is char
identifier is b123

keyword is char
identifier is c

keyword is if
identifier is sum
operator is =
operator is =
numbers is 10

identifier is printf
```

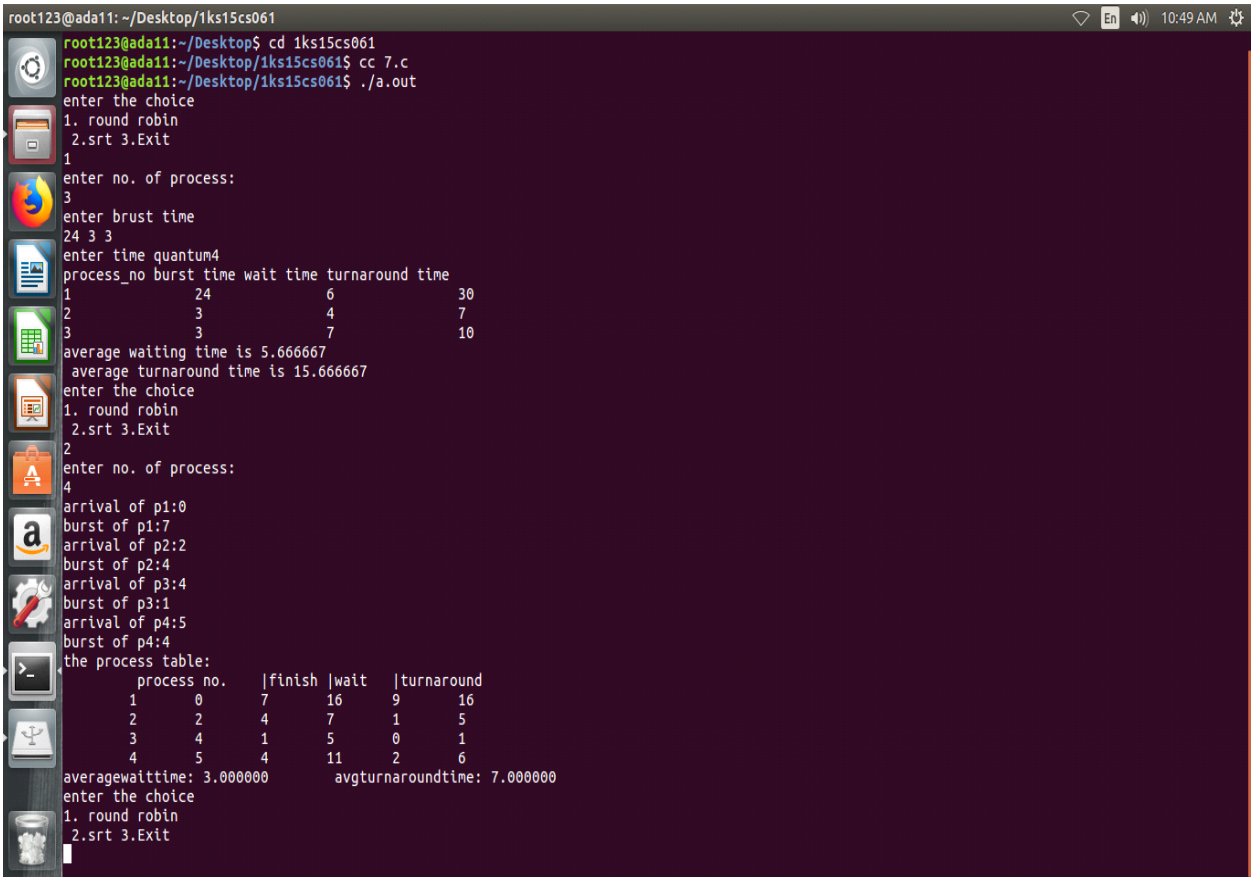
7. Design, develop and implement a C/C++/Java program to simulate the working of ***Shortest remaining time*** and ***Round Robin (RR)*** scheduling algorithms. Experiment with different quantum sizes for RR
algorithm.

```
#include<stdio.h>
#include<stdlib.h>
int arrival[10];
int burst[10];
int rem[10];
int wait[10];
int finish[10];
int turnaround[10];
int flag[10];
void roundrobin(int,int,int[],int[]);
void srtf(int);

int main()
{
int n,tq,choice;
int bt[10],st[10],i,j;
for(;;)
{
printf("enter the choice\n1. round robin\n 2.srt 3.Exit\n");
scanf("%d",&choice);
switch(choice)
{
case 1:
printf("enter no. of process:\n");
scanf("%d",&n);
printf("enter brust time\n");
for(i=0;i<n;i++)
{
  scanf("%d",&bt[i]);
  st[i]=bt[i];
}
printf("enter time quantum");
scanf("%d",&tq);
roundrobin(n,tq,st,bt);
break;

case 2:
printf("enter no. of process:\n");
scanf("%d",&n);
srtf(n);
break;
```

```
case 3:return 0;
}
}
}
void roundrobin(int n,int tq,int st[],int bt[])
{
int time=0;
int tat[10],wt[10],i,count=0,swt=0,stat=0,temp1,sq=0;

while(1)
{
for(i=0,count=0;i<n;i++)
{
temp1=tq;
if(st[i]==0)
{
count++;
continue;
}
if(st[i]>tq)
  st[i]=st[i]-tq;
else
if(st[i]>=0)
{
temp1=st[i];
st[i]=0;
}

sq=sq+temp1;
tat[i]=sq;
}

if(n==count)
break;
}
for(i=0;i<n;i++)
{
wt[i]=tat[i]-bt[i];
swt=swt+wt[i];
stat=stat+tat[i];
}
printf("process_no burst time wait time turnaround time\n");
for(i=0;i<n;i++)
   printf("%d\t\t%d\t\t%d\t\t%d\n",i+1,bt[i],wt[i],tat[i]);
printf("average waiting time is %f\n average turnaround time
is %f\n",(float)swt/n,(float)stat/n);
```

```
}

void srtf(int n)
{
int stat=0,swt=0,time=0,count=0,i,j,min=999;
for(i=1;i<=n;i++)
{
printf("arrival of p%d:",i);
scanf("%d",&arrival[i]);
printf("burst of p%d:",i);
scanf("%d",&burst[i]);
rem[i]=burst[i];
flag[i]=0;
}

while(1)
{

for(i=1,min=999;i<=n;i++)
  if(arrival[i]<=time&&flag[i]==0)
     if(rem[i]<min)
       {
         min=rem[i];
         j=i;
       }
        time++;
        rem[j]-=1;
       if(rem[j]==0)
       {
         finish[j]=time;
         flag[j]=1;
         count++;
       }
       if(count==n)
         break;
}
for(i=1;i<=n;i++)
 {
   turnaround[i]=finish[i]-arrival[i];
   wait[i]=turnaround[i]-burst[i];
   stat+=turnaround[i];
   swt+=wait[i];
 }

printf("the process table:\n\t process
no.\t|finish\t|wait\t|turnaround\t\n");
for(i=1;i<=n;i++)
```

```
   printf("\t%d  \t%d  \t%d  \t%d  \t%d
\t%d\n",i,arrival[i],burst[i],finish[i],wait[i],turnaround[i]);
printf("averagewaittime: %f\t
avgturnaroundtime: %f\n",(float)swt/n,(float)stat/n);
return;
}
```



8. Design, develop and implement a C/C++/Java program to implement ***Banker's algorithm***. Assume suitable input required to demonstrate the results.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
  int Max[10][10], need[10][10], alloc[10][10],
    avail[10], completed[10], safeSequence[10];
  int p, r, i, j, process, count;
  count = 0;

  printf("Enter the no of processes : ");
  scanf("%d", &p);

  for(i = 0; i< p; i++)
  completed[i] = 0;

  printf("\n\nEnter the no of resources : ");
  scanf("%d", &r);

  printf("\n\nEnter the Max Matrix for each process : ");
  for(i = 0; i < p; i++)
  {
    printf("\nFor process %d : ", i + 1);
    for(j = 0; j < r; j++)
    scanf("%d", &Max[i][j]);
}
 printf("\n\nEnter the allocation for each process : ");
 for(i = 0; i < p; i++)
 {
   printf("\nFor process %d : ",i + 1);
   for(j = 0; j < r; j++)
   scanf("%d", &alloc[i][j]);
 }

 printf("\n\nEnter the Available Resources : );
 for(i = 0; i < r; i++)
  scanf("%d", &avail[i]);

   for(i = 0; i < p; i++)
   for(j = 0; j < r; j++)
   need[i][j] = Max[i][j] - alloc[i][j];
```

```
do
{
  printf("\n Max matrix:\tAllocation matrix:\n");

  for(i = 0; i < p; i++)
  {
      for( j = 0; j < r; j++)
      printf("%d ", Max[i][j]);
      printf("\t\t");
      for( j = 0; j < r; j++)
      printf("%d ", alloc[i][j]);
      printf("\n");
  }

  process = -1;

  for(i = 0; i < p; i++)
  {
    if(completed[i] == 0)//if not completed
    {
      process = i ;
      for(j = 0; j < r; j++)
      {
        if(avail[j] < need[i][j])
        {
          process = -1;
          break;
              }
       }
     }
     if(process != -1)
       break;
   }

   if(process != -1)
   {
     printf("\nProcess %d runs to completion!", process + 1);
     safeSequence[count] = process + 1; count++;
     for(j = 0; j < r; j++)
     {
       avail[j] += alloc[process][j];
       alloc[process][j] = 0;
       Max[process][j] = 0;
       completed[process] = 1;
     }
```

```
      }
    }
    while(count != p && process != -1);

    if(count == p)
    {
      printf("\nThe system is in a safe state!!\n");
      printf("Safe Sequence : < ");
      for( i = 0; i < p; i++)
        printf("%d ", safeSequence[i]);
      printf(">\n");
    }
    else
      printf("\nThe system is in an unsafe state!!");

}
```

Output:

```
Enter the no of processes : 5
Enter the no of resources : 3
Enter the Max Matrix for each process :
For process 1 : 7
5
3
For process 2 : 3
2
2
For process 3 : 7
0
2
For process 4 : 2
2
2
For process 5 : 4
3
3

Enter the allocation for each process :
For process 1 : 0
1
0
For process 2 : 2
0
0
For process 3 : 3
```

```
0
2
For process 4 : 2
1
1
For process 5 : 0
0
2
Enter the Available Resources : 3
3
2
 Max matrix:   Allocation matrix:
7 5 3          0  1 0
3 2 2          2  0 0
7 0 2          3  0 2
2 2 2          2  1 1
4 3 3          0  0 2

Process 2 runs to completion!
 Max matrix:   Allocation matrix:
7 5 3          0  1 0
0 0 0          0  0 0
7 0 2          3  0 2
2 2 2          2  1 1

4 3 3           0 0 2

Process 3 runs to completion!
 Max matrix:     Allocation matrix:
7 5 3          0 1 0
0 0 0          0 0 0
0 0 0          0 0 0
2 2 2          2 1 1
4 3 3          0 0 2

Process 4 runs to completion!
 Max matrix:     Allocation matrix:
7 5 3          0 1 0
0 0 0          0 0 0
0 0 0          0 0 0
0 0 0          0 0 0
4 3 3          0 0 2

Process 1 runs to completion!
 Max matrix:     Allocation matrix:
0 0 0          0 0 0
0 0 0          0 0 0
```

```
0 0 0       0 0 0
0 0 0       0 0 0
4 3 3       0 0 2
```

Process 5 runs to completion!
The system is in a safe state!!
Safe Sequence: < 2 3 4 1 5 >

9. Design, develop and implement a C/C++/Java program to implement **page replacement algorithms LRU** and **FIFO**. Assume suitable input required to demonstrate the results.

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

void FIFO()
{
     char s[200];
     char F[200];
     int l,f,i,j=0,k,flag=0,cnt=0;

     printf("\nEnter the number of frames : ");
     scanf("%d",&f);

     printf("\nEnter the length of the string: ");
     scanf("%d",&l);

     printf("\nEnter the string: ");
     scanf("%s", s);

     for(i=0;i<f;i++)
     F[i]=' ';

     printf("\n\tPAGE\t\tFRAMES\t\t\tFAULTS");
     for(i=0;i<l;i++)
     {
               for(k=0;k<f;k++)
               if(F[k]==s[i])
               flag=1;
```

```
                if(flag==0)
                {
                     printf("\n\t%c\t",s[i]);
                     F[j]=s[i];
                     j++;
                     for(k=0;k<f;k++)
                     printf("\t%c",F[k]);
                     printf("\tPage-fault%d",cnt);
                     cnt++;
                }
                else
                {
                     flag=0;
                     printf("\n\t%c\t",s[i]);
                     for(k=0;k<f;k++)
                     printf("\t%c",F[k]);
                     printf("\tNo page-fault");
                }
                if(j==f)
                j=0;
     }
}

int findLRU(int time[], int n)
{
     int i, minimum = time[0], pos = 0;
     for(i = 1; i < n; ++i)
     {
          if(time[i] < minimum)
          {
               minimum = time[i];
               pos = i;
          }
     }
     return pos;
}

int lru()
{
```

```
int no_of_frames, no_of_pages, frames[10], counter = 0;
int time[10], flag1, flag2, i, j, pos, faults = 0, page;
char s[200];

printf("\nEnter number of frames: ");
scanf("%d", &no_of_frames);

printf("\nEnter number of pages: ");
scanf("%d", &no_of_pages);

printf("\nEnter reference string: ");
scanf("%s", s);

for(i = 0; i < no_of_frames; ++i)
frames[i] = -1;

for(i = 0; i < no_of_pages; ++i)
{
     flag1 = flag2 = 0;
     page = s[i] - '0';
     for(j = 0; j < no_of_frames; ++j)
     {
          if(frames[j] == page)
          {
               counter++;
               time[j] = counter;
               flag1 = flag2 = 1;
               break;
          }
     }
     if(flag1 == 0)
     {
          for(j = 0; j < no_of_frames; ++j)
          {
               if(frames[j] == -1)
               {
                    counter++;
                    faults++;
                    frames[j] = page;
```

```
                        time[j] = counter;
                        flag2 = 1;
                        break;
                    }
                }
            }
            if(flag2 == 0)
            {
                pos = findLRU(time, no_of_frames);
                counter++;
                faults++;
                frames[pos] = page;
                time[pos] = counter;
            }
            printf("\n");
            for(j = 0; j < no_of_frames; ++j)
            printf("%d\t", frames[j]);
        }
        printf("\n\nTotal Page Faults = %d", faults);
        return 0;
}

int main()
{
        int ch,YN=1,i,l,f;
        char F[10],s[25];
        do
        {
            printf("\nOptions : ");
            printf("\n\n1:FIFO\n2:LRU \n3:EXIT");
            printf("\n\nEnter your choice: ");
            scanf("%d",&ch);
            switch(ch)
            {
                case 1:   FIFO();
                    break;
                case 2: lru();
                    break;
                default:
```

```
                    exit(0);
            }
            printf("\n\nPress 1 to continue.. 0 to exit ");
            scanf("%d",&YN);
        }while(YN==1);
        return(0);
}
```

```
root123@root123-Inspiron-N5010:~$ ./a.out

Options :

1:FIFO
2:LRU
3:EXIT

Enter your choice: 1

Enter the number of frames : 4

Enter the length of the string: 13

Enter the string: 2342137543231

        PAGE            FRAMES                  FAULTS
        2               2                               Page-fault0
        3               2       3                       Page-fault1
        4               2       3       4               Page-fault2
        2               2       3       4               No page-fault
        1               2       3       4       1       Page-fault3
        3               2       3       4       1       No page-fault
        7               7       3       4       1       Page-fault4
        5               7       5       4       1       Page-fault5
        4               7       5       4       1       No page-fault
        3               7       5       3       1       Page-fault6
        2               7       5       3       2       Page-fault7
        3               7       5       3       2       No page-fault
        1               1       5       3       2       Page-fault8

Press 1 to continue.. 0 to exit
```

```
Options :

1:FIFO
2:LRU
3:EXIT

Enter your choice: 2

Enter number of frames: 4

Enter number of pages: 13

Enter reference string: 2342137543231

2       -1      -1      -1
2       3       -1      -1
2       3       4       -1
2       3       4       -1
2       3       4       1
2       3       4       1
2       3       7       1
5       3       7       1
5       3       7       4
5       3       7       4
5       3       2       4
5       3       2       4
1       3       2       4

Total Page Faults = 9

Press 1 to continue.. 0 to exit
```