SYSTEM SOFTWARE LABORATORY (Effective from the academic year 2018 -2019) SEMESTER – VI			
Course Code	**18CSL66**	**CIE Marks**	40
Number of Contact Hours/Week	0:2:2	**SEE Marks**	60
Total Number of Lab Contact Hours	36	**Exam Hours**	03
Credits – 2			

Course Learning Objectives: This course (18CSL66) will enable students to:

- To make students familiar with Lexical Analysis and Syntax Analysis phases of Compiler Designand implement programs on these phases using LEX & YACC tools and/or C/C++/Java
- To enable students to learn different types of CPU scheduling algorithms used in operatingsystem.
- To make students able to implement memory management - page replacement and deadlockhandling algorithms

Descriptions (if any):

Exercises to be prepared with minimum three files (Where ever necessary):

1. Header file.
2. Implementation file.
3. Application file where main function will be present.

The idea behind using three files is to differentiate between the developer and user sides. In the developer side, all the three files could be made visible. For the user side only header file and application files could be made visible, which means that the object code of the implementation file could be given to the user along with the interface given in the header file, hiding the source file, if required. Avoid I/O operations (printf/scanf) and use ***data input file*** where ever it is possible.

Programs List:

Installation procedure of the required software must be demonstrated, carried out in groups anddocumented in the journal.

1.	
a.	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.
b.	Write YACC program to evaluate ***arithmetic expression*** involving operators: +, -, *, and /
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)
3.	Design, develop and implement YACC/C program to construct ***Predictive / LL(1) Parsing Table*** for the grammar rules: $A \rightarrow aBa$, $B \rightarrow bB \mid \varepsilon$. Use this table to parse thesentence: ***abba$***
4.	Design, develop and implement YACC/C program to demonstrate ***Shift Reduce Parsing*** technique for the grammar rules: $E \rightarrow E+T \mid T$, $T \rightarrow T*F \mid F$, $F \rightarrow (E) \mid id$ and parse the sentence: ***id + id * id***.

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***
6.	
a.	Write a LEX program to eliminate comment lines in a C program and copy the resulting program into a separate file.
b.	Write YACC program to recognize valid identifier, operators and keywords in the given text (C program) file.
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.
8.	Design, develop and implement a C/C++/Java program to implement Banker's algorithm. Assume suitable input required to demonstrate the results
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.

Laboratory Outcomes: The student should be able to:

- Implement and demonstrate Lexer's and Parser's
- Evaluate different algorithms required for management, scheduling, allocation andcommunication used in operating system.

Conduct of Practical Examination:

- Experiment distribution
  - For laboratories having only one part: Students are allowed to pick one experiment fromthe lot with equal opportunity.
  - For laboratories having PART A and PART B: Students are allowed to pick one experiment from PART A and one experiment from PART B, with equal opportunity.
- Change of experiment is allowed only once and marks allotted for procedure to be made zero ofthe changed part only.
- Marks Distribution *(Courseed to change in accoradance with university regulations)*
  m) For laboratories having only one part – Procedure + Execution + Viva-Voce: 15+70+15 = 100 Marks
  n) For laboratories having PART A and PART B
    i. Part A – Procedure + Execution + Viva = 6 + 28 + 6 = 40 Marks
    ii. Part B – Procedure + Execution + Viva = 9 + 42 + 9 = 60 Marks

1. **a) 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-zA-Z][0-9A-Za-z]*  {id++; printf("\n Identifier:"); ECHO;}
[\+\-\*\/\=]  {op++; printf("\n Operartor:"); ECHO;}
 "("  {v++;}
 ")"  {v--;}
 ";"  {flag=1;}
 .|\n  {;}
 %%
main()
 {
    printf("Enter the expression");
     yylex();
     if((op+1) ==id && v==0 && flag==0)
            printf("\n Expression is Valid\n");
     else
             printf("\n Expression is Invalid\n");
    printf("no. of identifiers=%d\n",id);
    printf("no. of operators=%d\n",op)
  }
```

Execution Steps:

Lex <lexfilename.l>
cc lex.yy.c –ll
. /a.out

Output:

```
admin1@admin1-HP-ProDesk-400-G3-DM:~$ ./a.out
Enter the expressiona+b

 Identifier:a
 Operartor:+
 Identifier:b

 Expression is Valid
admin1@admin1-HP-ProDesk-400-G3-DM:~$ ./a.out
Enter the expression(a+b)-c*d+(e/f)

 Identifier:a
 Operartor:+
 Identifier:b
 Operartor:-
 Identifier:c
 Operartor:*
 Identifier:d
 Operartor:+
 Identifier:e
 Operartor:/
 Identifier:f

 Expression is Valid
```

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;}
[\+\-\*\/]   {return yytext[0];}
[)]  {return yytext[0];}
[(]  {return yytext[0];}
.   { ; }
\n   {return 0;}
%%
```

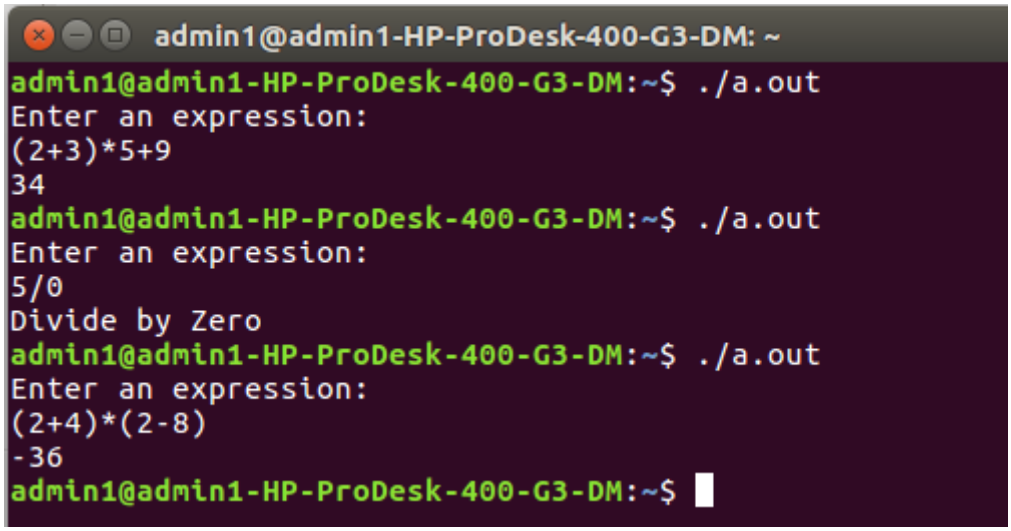
YACC Part

```
%{
#include<stdio.h>
#include<stdlib.h>
%}
 %token  num
%left  '+' '-'
%left  '*' '/'
 %%
S: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();
 }
```

Execution Steps:

YACC –d <yaccfilename.y>
lex <lexfilename.l>
cc y.tab.c lex.yy.c –ll
./a.out

Output:



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);
}
```

Execution Steps:

```
YACC –d <yaccfilename.y>
lex <lexfilename.l>
cc y.tab.c lex.yy.c –ll
/a.out
```

Output:


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
aaaaaaaaaaaaaaaab
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<stdio.h>
#include<string.h>
char prod[3][15]={"A->aBa","B->bB","B->@"};
char table[2][3][3]={{"aBa"," "," "}, {"@","bB"," "}};
int size[2][3]={3,0,0,1,2,0},n;
char s[20],stack[20];
void display(int i,int j)
{
    int k;
    for(k=0;k<=i;k++)
            printf("%c",stack[k]);
    printf("\t");
    for(k=j;k<n;k++)
            printf("%c",s[k]);
    printf("\n");
}

int main()
{
    int i,j,k,row,col,flag=0;
    printf("\n the grammar is:\n");
    for(i=0;i<3;i++)
            printf("%s\n",prod[i]);
    printf("\n predicting parsing table is\n\n");
    printf("\ta\tb\t$\n");
    printf("----------\n");
    for(i=0;i<2;i++)
    {
            if(i==0)printf("A");
            else printf("\nB");
            for(j=0;j<3;j++)
            {
                    printf("\t%s",table[i][j]);
            }
    }
    printf("\n enter the input string:");
    scanf("%s",s);
    printf("\n stack input");
```

```
printf("\n-----------------------\n");
printf("$A\t%s$\n",s);
strcat(s,"$");
n=strlen(s);
stack[0]='$';
stack[1]='A';
i=1;
j=0;
while(1)
{
        if(stack[i]==s[j])
        {
                i--;
                j++;
                if(stack[i]=='$' && s[j]=='$')
                {
                        printf("$ $\n SUCCESS\n");
                        break;
                }
                else
                        if(stack[i]=='$' && s[j]!='$')
                        {
                                printf("ERROR\n");
                                break;
                        }
                display(i,j);
        }
        switch(stack[i])
        {
                case 'A':row=0;
                        break;
                case'B':row=1;
                        break;
        }
        switch(s[j])
        {
                case 'a':col=0;
                        break;
                case 'b':col=1;
                        break;
                case '$':col=2;
```

```
                        break;
                }
                if(table[row][col][0]=='\0')
                {
                        printf("\n ERROR\n");
                        break;
                }
                else if(table[row][col][0]=='@')
                {
                        i--;
                        display(i,j);
                }
                else
                {
                        for(k=size[row][col]-1;k>=0;k--)
                        {
                                stack[i]=table[row][col][k];
                                i++;
                        }
                        i--;
                        display(i,j);
                }
        }
    }
```

Execution Steps:

cc filename.c

./a.out

Output:

```
rit@rit: ~
rit@rit:~$ cc 3.c
rit@rit:~$ ./a.out

 the grammar is:
A->aBa
B->bB
B->@

 predicting parsing table is

        a       b       $
----------------
A       aBa
B       @       bB
 enter the input string:abba

 stack input
-------------------------
$A      abba$
$aBa    abba$
$aB     bba$
$aBb    bba$
$aB     ba$
$aBb    ba$
$aB     a$
$a      a$
$ $
 SUCCESS
rit@rit:~$
```

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<string.h>
#include<stdio.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()
{
    printf("GRAMMAR is E->E+T|T \n T->T*F|F \n F->(E)|id \n");
    printf("enter input string ");
    scanf("%s",a);
    c=strlen(a);
    strcpy(act,"SHIFT->");
    printf("stack \t  input \t  action\n");
    printf("----------------------------------------");
    printf("\n$\t%s\t",a);
    for(k=0,i=0; j<c; k++,i++,j++)
    {
        if(a[j]=='i' && a[j+1]=='d')
        {
            stk[i]=a[j];
            stk[i+1]=a[j+1];
            stk[i+2]='\0';
            a[j]=' ';
            a[j+1]=' ';
            printf("\n$%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();
        }
    }
}
```

```
void check()
{
    strcpy(ac,"REDUCE TO ");
    for(z=0; z<c; z++)
        if(stk[z]=='i' && stk[z+1]=='d')
        {
            stk[z]='F';
            stk[z+1]='\0';
            printf("\n$%s\t%s$\t%sF",stk,a,ac);
            j++;
        }

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

    for(z=0; z<c; z++)
        if(stk[z]=='(' && stk[z+1]=='E' && stk[z+2]==')')
        {
            stk[z]='F';
            stk[z+1]='\0';
            stk[z+1]='\0';
            printf("\n$%s\t%s$\t%sF",stk,a,ac);
            i=i-2;
        }
    for(z=0;z<c;z++)
    {
        if(stk[z]=='E' && stk[z+1]=='+' && stk[z+2]=='T' && stk[z+3]=='*')
        break;
```

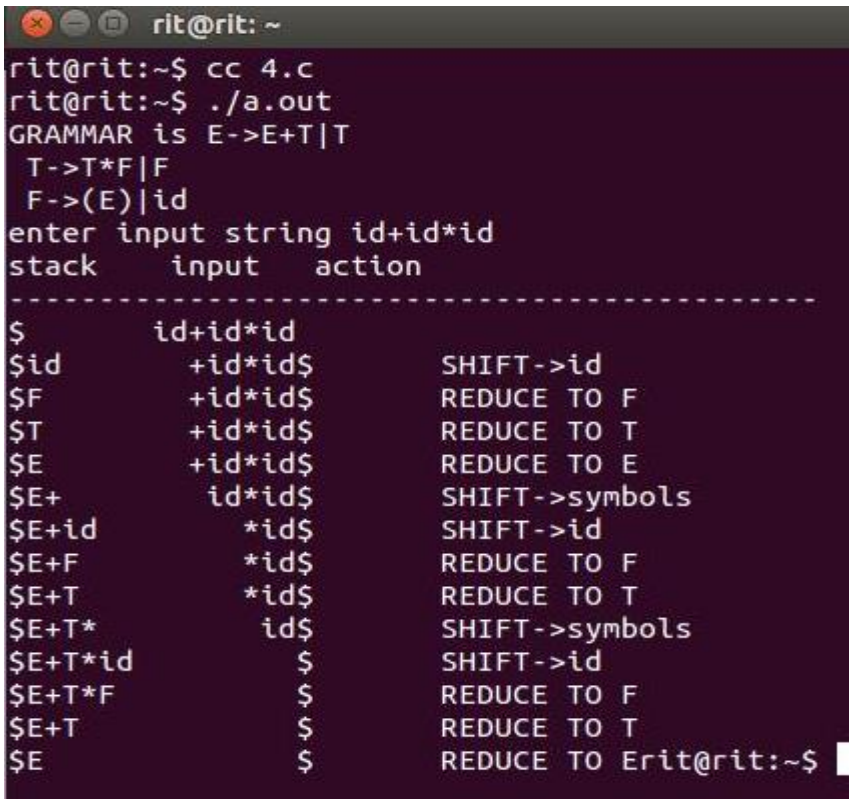
```
            if(stk[z]=='E' && stk[z+1]=='+' && stk[z+2] == 'T')
                    if(a[j+1]=='*')
                            break;
                    else
                    {
                            stk[z]='E';
                            stk[z+1]='\0';
                            stk[z+2]='\0';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

Execution Steps:

cc filename.c

./a.out

Output:



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>
char op[2],arg1[5],arg2[5],result[5];
int count;
void main()
{
    FILE *fp1,*fp2;
    fp1=fopen("input.txt","r");
    fp2=fopen("output.txt","w");
    while(!feof(fp1)&&count<4)
    {
            count++;
            fscanf(fp1,"%s%s%s%s",result,arg1,op,arg2);
            if(strcmp(op,"+")==0)
            {
                    fprintf(fp2,"\nMOV R0,%s",arg1);
                    fprintf(fp2,"\nADD R0,%s",arg2);
                    fprintf(fp2,"\nMOV %s,R0",result);
            }
            if(strcmp(op,"*")==0)
            {
                    fprintf(fp2,"\nMOV R0,%s",arg1);
                    fprintf(fp2,"\nMUL R0,%s",arg2);
                    fprintf(fp2,"\nMOV %s,R0",result);
            }
            if(strcmp(op,"-")==0)
            {
                    fprintf(fp2,"\nMOV R0,%s",arg1);
                    fprintf(fp2,"\nSUB R0,%s",arg2);
                    fprintf(fp2,"\nMOV %s,R0",result);
            }
```

```
                if(strcmp(op,"/")==0)
                {
                        fprintf(fp2,"\nMOV R0,%s",arg1);
                        fprintf(fp2,"\nDIV R0,%s",arg2);
                        fprintf(fp2,"\nMOV %s,R0",result);
                }
                if(strcmp(op,"=")==0)
                {
                        fprintf(fp2,"\nMOV R0,%s",arg1);
                        fprintf(fp2,"\nMOV %s,R0",result);
                }
        }
    fclose(fp1);
    fclose(fp2);
}
```

Execution Steps:

cc filename.c

./a.out

Output:

input.txt

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

output.txt

```
MOV R0,-B
MOV T1,R0
MOV R0,C
ADD R0,D
MOV T2,R0
MOV R0,T1
MUL R0,T2
MOV T3,R0
MOV R0,T3
MOV 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.

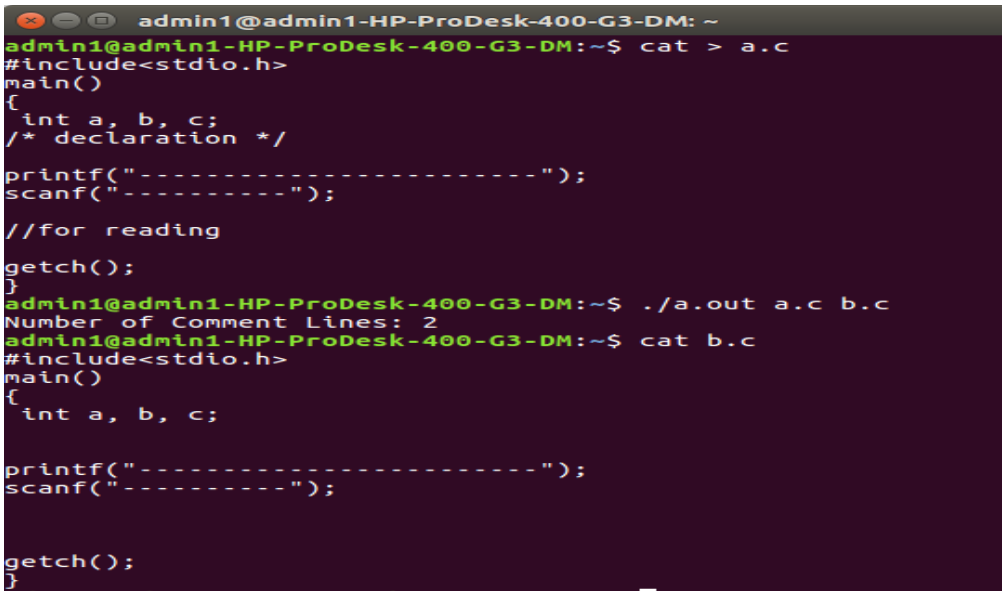
```
%{
#include<stdio.h>
int c_count=0;
%}

%%
"/*"[^*/].*"*/"  { c_count++; }
"//".*   { c_count++; }
%%
int main( int argc, char **argv)
{
    if(argc>1)
    {
            yyin=fopen(argv[1],"r");
            yyout=fopen(argv[2],"w");
            yylex();
            printf("Number of Comment Lines: %d\n",c_count);
    }
return 0;
}
```

Execution Steps:

Lex <lexfilename.l>
cc lex.yy.c –ll
. /a.out a.c b.c

Output:



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-Z][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++;}
        ;
%%
```

```
main()
{
        yyin = fopen("input.c", "r");
            do {
                yyparse();
                } while (!feof(yyin));
        printf("numbers = %d\nKeywords = %d\nIdentifiers = %d\noperators =
                                                        %d\n", dig, key,id, op);
}
int yyerror() {
printf("EEK, parse error! Message: ");
exit(-1);
}
```

Execution Steps:

```
YACC –d <yaccfilename.y>
lex <lexfilename.l>
cc y.tab.c lex.yy.c –ll
./a.out
```

Output:

Input file

```
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
identifier is pass

keyword is else

identifier is printf
identifier is fail


numbers = 1
Keywords = 7
Identifiers = 10
operators = 2
admin1@admin1-HP-ProDesk-400-G3-DM:~$
```

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>
int main()
{
    int count,j,n,time,flag=0,tq,ch=0;
    int wait_time=0,turnaround_time=0,at[10],bt[10],rt[10];
    int endTime,i,smallest;
    int remain=0,sum_wait=0,sum_turnaround=0;
    printf("1.Round Robin\n2.SRTF\n");
    scanf("%d",&ch);
    printf("Enter number of processes:");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("Enter arrival time for process p %d:",i+1);
        scanf("%d",&at[i]);
        printf("enter burst time for process p %d:",i+1);
        scanf("%d",&bt[i]);
        rt[i]=bt[i];
    }
    switch(ch)
    {
        case 1:printf("Enter Time Quantum:\t");
            scanf("%d",&tq);
            remain=n;
            printf("\n Process time|turnaround Time|waiting Time\n");
            for(time=0,count=0;remain!=0;)
            {
                if(rt[count]<=tq && rt[count]>0)
                {
                    time +=rt[count];
                    rt[count]=0;
                    flag=1;
                }
                else if(rt[count]>0)
                {
                    rt[count]-=tq;
                    time +=tq;
                }
                if(rt[count]==0 && flag==1)
                {
                    remain--;
                    printf("p[%d]\t|\t%d\t|\t%d\n",count+1,time-
                                at[count],time-at[count]-bt[count]);
                    wait_time +=time-at[count]-bt[count];
```

```
                        turnaround_time +=time-at[count];
                        flag=0;
                    }
                    if(count==n-1)
                        count=0;
                    else if(at[count+1]<=time)
                        count++;
                    else count=0;
                }
                printf("\n Average waiting time=%0.2f\n",wait_time*1.0/n);
                printf("Avg Turnaround time=%0.2f\n",turnaround_time*1.0/n);
                break;

        case 2: remain=0;
                printf("\n process|Turnaround Time|waiting time\n");
                rt[9]=9999;
                for(time=0;remain!=n;time++)
                {
                    smallest=9;
                    for(i=0;i<n;i++)
                        if(at[i]<=time && rt[i]<rt[smallest] &&
                            rt[i]>0)
                        smallest=i;
                    rt[smallest]--;
                    if(rt[smallest]==0)
                    {
                        remain++;
                        endTime=time+1;
                        printf("\np[%d]\t|\t%d\t|\t%d", smallest+1,
                            endTime-at[smallest],endTime-bt[smallest]-
                                                        at[smallest]);
                        printf("\n");
                        sum_wait +=endTime-bt[smallest]-at[smallest];
                        sum_turnaround +=endTime-at[smallest];
                    }
                }
                printf("\n Average waiting time=%f\n",sum_wait*1.0/n);
                printf("Average Turnaround time=%f\n",sum_turnaround*1.0/n);
                break;
    }
    return 0;
}
```

Execution Steps:

cc filename.c

./a.out

Output:

```
rit@rit: ~
rit@rit:~$ cc 7.c
rit@rit:~$ ./a.out
1.Round Robin
2.SRTF
1
Enter number of processes:4
Enter arrival time for process p 1:0
enter burst time for process p 1:9
Enter arrival time for process p 2:1
enter burst time for process p 2:5
Enter arrival time for process p 3:2
enter burst time for process p 3:3
Enter arrival time for process p 4:3
enter burst time for process p 4:4
Enter Time Quantum:     4

 Process time|turnaround Time|waiting Time
p[3]     |        9        |        6
p[4]     |        12       |        8
p[2]     |        19       |        14
p[1]     |        21       |        12

 Average waiting time=10.00
Avg Turnaround time=15.25
rit@rit:~$ ./a.out
```

```
rit@rit:~$ ./a.out
1.Round Robin
2.SRTF
2
Enter number of processes:5
Enter arrival time for process p 1:2
enter burst time for process p 1:1
Enter arrival time for process p 2:1
enter burst time for process p 2:5
Enter arrival time for process p 3:4
enter burst time for process p 3:1
Enter arrival time for process p 4:0
enter burst time for process p 4:6
Enter arrival time for process p 5:2
enter burst time for process p 5:3

 process|Turnaround Time|waiting time

p[1]     |        1        |        0

p[3]     |        1        |        0

p[5]     |        5        |        2

p[2]     |        10       |        5

p[4]     |        16       |        10

 Average waiting time=3.400000
Average Turnaround time=6.600000
rit@rit:~$
```

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 number of process:");
        scanf("%d",&p);
        for(i=0;i<p;i++)
                completed[i]=0;
        printf("\n\nEnter the number of resources:");
        scanf("%d",&r);
        printf("\n\nEnter 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\n Enter the allocation for each process:");
        for(i=0;i<p;i++)
        {
                printf("\n For process %d:",i+1);
                for(j=0;j<r;j++)
                        scanf("%d",&alloc[i][j]);
        }
        printf("\n\n Enter 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 MaxMatrix:\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)
                        {
                                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("\n process %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("\n The system is in safe state!\n");
                printf("Safe Sequence:<");
                for(i=0;i<p;i++)
                        printf("%d",safesequence[i]);
                printf(">\n");
        }
```

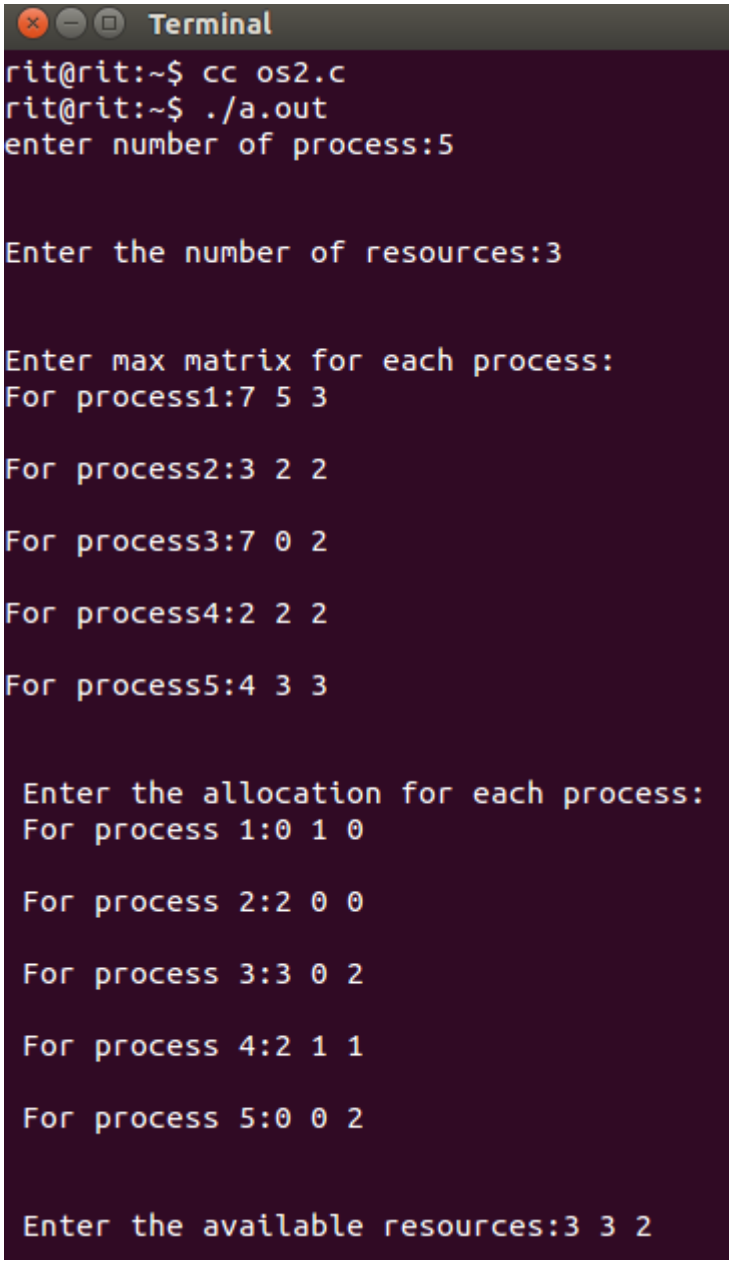
```
                else
                        printf("\n The system is in an unsafe state!");
                return 0;
        }
```

Execution Steps:

cc filename.c

./a.out

Output:



```
 Enter the available resources:3 3 2

 MaxMatrix:     Allocation Matrix:
753             010
322             200
702             302
222             211
433             002

 process 2 runs to completion!
 MaxMatrix:     Allocation Matrix:
753             010
000             000
702             302
222             211
433             002

 process 3 runs to completion!
 MaxMatrix:     Allocation Matrix:
753             010
000             000
000             000
222             211
433             002

 process 4 runs to completion!
 MaxMatrix:     Allocation Matrix:
753             010
000             000
000             000
000             000
433             002

 process 1 runs to completion!
 MaxMatrix:     Allocation Matrix:
000             000
000             000
000             000
000             000
433             002

 process 5 runs to completion!
 The system is in safe state!
Safe Sequence:<23415>
rit@rit:~$
```

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 [ ],char [ ],int,int);
void lru(char [ ],char [ ],int,int);
void opt(char [ ],char [ ],int,int);
int main()
{
        int ch,YN=1,i,l,f;
        char F[10],s[25];
        printf("\n\n\tEnter the no of empty frames: ");
        scanf("%d",&f);
        printf("\n\n\tEnter the length of the string: ");
        scanf("%d",&l);
        printf("\n\n\tEnter the string: ");
        scanf("%s",s);
        for(i=0;i<f;i++)
                F[i]=-1;
        do
        {
                printf("\n\n\t********** MENU **********");
                printf("\n\n\t1:FIFO\n\n\t2:LRU\n\n\t3:EXIT");
                printf("\n\n\tEnter your choice:");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1: for(i=0;i<f;i++)
                                {
                                        F[i]=-1;
                                }
                                FIFO(s,F,l,f);
                                break;
                        case 2: for(i=0;i<f;i++)
                                {
                                        F[i]=-1;
                                }
                                lru(s,F,l,f);
                                break;
                        case 3: exit(0);
                 }
                printf("\n\n\tDo u want to continue IF YES PRESS 1\n\n\t IF
                                                        NO PRESS 0 :");
                scanf("%d",&YN);
        }while(YN==1);
        return(0);
}
```

```
void FIFO(char s[],char F[],int l,int f)
{
		int i,j=0,k,flag=0,cnt=0;
		printf("\n\tPAGE\tFRAMES\t FAULTS");
		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(" %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(" %c",F[k]);
				}
				printf("\tNo page-fault");
			}
			if(j==f)
				j=0;
		}
}

void lru(char s[],char F[],int l,int f)
{
		int i,j=0,k,m,flag=0,cnt=0,top=0;
		printf("\n\tPAGE\t FRAMES\t FAULTS");
		for(i=0;i<l;i++)
		{
			for(k=0;k<f;k++)
			{
```

```
                        if(F[k]==s[i])
                        {
                                flag=1;
                                break;
                        }
                }
                printf("\n\t%c\t",s[i]);
                if(j!=f && flag!=1)
                {
                        F[top]=s[i];
                        j++;
                        if(j!=f)
                                top++;
                }
                else
                {
                        if(flag!=1)
                        {
                                for(k=0;k<top;k++)
                                {
                                        F[k]=F[k+1];
                                }
                                F[top]=s[i];
                        }
                        if(flag==1)
                        {
                                for(m=k;m<top;m++)
                                {
                                        F[m]=F[m+1];
                                }
                                F[top]=s[i];
                        }
                }
                for(k=0;k<f;k++)
                {
                        printf(" %c",F[k]);
                }
                if(flag==0)
                {
                        printf("\tPage-fault%d",cnt);
                        cnt++;
                }
                else
                        printf("\tNo page fault");
                flag=0;
        }
}
```

Execution Steps:

cc filename.c

./a.out

Output:

```
rit@rit:~$ cc oss3.c
rit@rit:~$ ./a.out


        Enter the no of empty frames: 3


        Enter the length of the string: 5


        Enter the string: hello


        *********** MENU ***********

        1:FIFO

        2:LRU

        4:EXIT

        Enter your choice:1

        PAGE    FRAMES   FAULTS
        h        h ◆ ◆  Page-fault0
        e        h e ◆  Page-fault1
        l        h e l  Page-fault2
        l        h e l  No page-fault
        o        o e l  Page-fault3

        Do u want to continue IF YES PRESS 1

        IF NO PRESS 0 :1


        *********** MENU ***********

        1:FIFO

        2:LRU

        4:EXIT

        Enter your choice:2
```

```
********** MENU **********

1:FIFO

2:LRU

4:EXIT

Enter your choice:2

PAGE      FRAMES  FAULTS
h         h ♦ ♦   Page-fault0
e         h e ♦   Page-fault1
l         h e l   Page-fault2
l         h e l   No page fault
o         e l o   Page-fault3

Do u want to continue IF YES PRESS 1

IF NO PRESS 0 :
```