



**Gopalan College**  
of Engineering and Management  
ISO 9001 : 2008

**APPROVED BY AICTE NEW DELHI, AFFILIATED TO VTU BELGAUM**



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**SYSTEM SOFTWARE AND OPERATING SYSTEM  
LAB MANUAL - 10CSL58  
SEMESTER-V  
2016-2017**

**Prepared by:**  
Arvind R  
Assistant professor  
Dept. of CSE  
GCEM

**Reviewed by:**  
Suparna K  
Assistant Professor  
Dept. of CSE  
GCEM

**Approved by:**  
Dr. A.A. Powly Thomas  
Principal  
GCEM

181/1, 182/1, Hoodi Village, Sonnenahalli, K.R. Puram, Bengaluru,  
Karnataka-560 048.

## TABLE OF CONTENTS

<b>S. No</b>	<b>Title of Contents</b>	<b>Page</b>	
		<b>From</b>	<b>To</b>
<b>1</b>	<b>Syllabus</b>	<b>3</b>	<b>4</b>
<b>2</b>	<b>Course Objective and Course Outcome</b>	<b>5</b>	<b>5</b>
<b>3</b>	<b>Do's And Don'ts</b>	<b>6</b>	<b>6</b>
<b>4</b>	<b>List of Experiments</b>	<b>7</b>	<b>7</b>
<b>5</b>	<b>Programs and Sample output</b>	<b>8</b>	<b>50</b>

**SYSTEM SOFTWARE & OPERATING SYSTEMS LABORATORY****Subject Code: 10CSL58****Semester : 5<sup>th</sup> CSE****Max Marks : 50****PART – A****LEX and YACC Programs:***Design, develop, and execute the following programs using LEX:*

- 1) a. Program to count the number of characters, words, spaces and lines in a given input file.  
b. Program to count the numbers of comment lines in a given C program. Also eliminate them and copy the resulting program into separate file.
- 2) a. Program to recognize a valid arithmetic expression and to recognize the identifiers and operators present. Print them separately.  
b. Program to recognize whether a given sentence is simple or compound.
- 3) Program to recognize and count the number of identifiers in a given input file.

*Design, develop, and execute the following programs using YACC:*

- 4) a. Program to recognize a valid arithmetic expression that uses operators +, -, \* and /.  
b. Program to recognize a valid variable, which starts with a letter, followed by any number of letters or digits.
- 5) a. Program to evaluate an arithmetic expression involving operators +, -, \* and /.  
b. Program to recognize strings 'aaab', 'abbb', 'ab' and 'a' using the grammar ( $a^n b^n$ ,  $n \geq 0$ ).
- 6) Program to recognize the grammar ( $a^n b$ ,  $n \geq 10$ ).

**PART- B****Unix Programming:***Design, develop, and execute the following programs:*

- 7) a. Non-recursive shell script that accepts any number of arguments and prints them in the Reverse order( For example, if the script is named rargs, then executing rargs A B C should produce C B A on the standard output).  
b. C program that creates a child process to read commands from the standard input and execute them (a minimal implementation of a shell – like program). You can assume that no arguments will be passed to the commands to be executed.
- 8) a. Shell script that accepts two file names as arguments, checks if the permissions for

these files are identical and if the permissions are identical, outputs the common permissions, otherwise outputs each file name followed by its permissions.

b. C program to create a file with 16 bytes of arbitrary data from the beginning and another 16 bytes of arbitrary data from an offset of 48. Display the file contents to demonstrate how the hole in file is handled.

9) a. Shell script that accepts file names specified as arguments and creates a shell script that contains this file as well as the code to recreate these files. Thus if the script generated by your script is executed, it would recreate the original files (This is same as the “bundle” script described by Brian W. Kernighan and Rob Pike in “The Unix Programming Environment”, Prentice – Hall India).

b. C program to do the following: Using `fork()` create a child process. The child process prints its own process-id and id of its parent and then exits. The parent process waits for its child to finish (by executing the `wait()`) and prints its own process-id and the id of its child process and then exits.

### **Operating Systems:**

10) Design, develop and execute a program in C / C++ to simulate the working of Shortest Remaining Time and Round-Robin Scheduling Algorithms. Experiment with different quantum sizes for the Round-Robin algorithm. In all cases, determine the average turn-around time. The input can be read from key board or from a file.

11) Using Open MP, Design, develop and run a multi-threaded program to generate and print Fibonacci Series. One thread has to generate the numbers up to the specified limit and another thread has to print them. Ensure proper synchronization.

12. Design, develop and run a program to implement the Banker’s Algorithm. Demonstrate its working with different data values.

### **Instructions:**

In the examination, a combination of one LEX and one YACC Program has to be asked from Part A for a total of 30 marks and one programming exercise from Part B has to be asked for a total of 20 marks.

## **COURSE OBJECTIVE:-**

- To acquire the implementation knowledge of System Software concepts and FAFL grammar concepts through UNIX supported tools Lex and Yacc.
- To acquire the knowledge of retrieval of the information in the static files and manipulate the contents of files using Unix utilities in scripting languages

## **COURSE OUTCOME:-**

After studying this course, students will be able to do:

- Implementation of file handling concepts and process concepts
- Implement the Concept of Parsers
- Implement the concepts of System software
- To provide an understanding of the design aspects of operating system
- 

## **DO'S AND DON'TS**

**Do's**

Do wear ID card and follow dress code.

- Do log off the computers when you finish.
- Do ask for assistance if you need help.
- Do keep your voice low when speaking to others in the LAB.
- Do ask for assistance in downloading any software.
- Do make suggestions as to how we can improve the LAB.
- In case of any hardware related problem, ask LAB in charge for solution.
- If you are the last one leaving the LAB, make sure that the staff in charge of the LAB is informed to close the LAB.
- Be on time to LAB sessions.
- Do keep the LAB as clean as possible.

**Don'ts**

- Do not use mobile phone inside the lab.
- Don't do anything that can make the LAB dirty (like eating, throwing waste papers etc).
- Do not carry any external devices without permission.
- Don't move the chairs of the LAB.
- Don't interchange any part of one computer with another.
- Don't leave the computers of the LAB turned on while leaving the LAB.
- Do not install or download any software or modify or delete any system files on any lab computers.
- Do not damage, remove, or disconnect any labels, parts, cables, or equipment.
- Don't attempt to bypass the computer security system.
- Do not read or modify other user's file.
- If you leave the lab, do not leave your personal belongings unattended. We are not responsible for any theft.

## LIST OF EXPERIMENTS

S. No	Title of the Experiment	Page	
		From	To
<b>PART-A</b>			
<b>1.a</b>	Counting the number of characters, words, spaces and lines	<b>7</b>	<b>11</b>
<b>1.b</b>	Counting the numbers of comment lines	<b>12</b>	<b>13</b>
<b>2.a</b>	Recognizing a valid arithmetic expression and to recognize valid identifiers	<b>14</b>	<b>14</b>
<b>2.b</b>	Recognizing whether a given sentence is simple or compound	<b>15</b>	<b>15</b>
<b>3</b>	Counting the number of identifiers	<b>16</b>	<b>17</b>
<b>4.a</b>	Counting a valid arithmetic expression	<b>19</b>	<b>20</b>
<b>4.b</b>	Recognizing a valid variable	<b>20</b>	<b>21</b>
<b>5.a</b>	Evaluating an arithmetic expression	<b>22</b>	<b>23</b>
<b>5.b</b>	Recognizing strings in a grammar	<b>24</b>	<b>24</b>
<b>6</b>	Recognizing a given grammar	<b>25</b>	<b>26</b>
<b>PART-B</b>			
<b>7.a</b>	Accepting any number of arguments and printing them in the Reverse order	<b>27</b>	<b>28</b>
<b>7.b</b>	Creating a child process	<b>29</b>	<b>30</b>
<b>8.a</b>	Checking the permission of a file	<b>31</b>	<b>33</b>
<b>8.b</b>	Displaying the file contents to demonstrate the hole in a file	<b>34</b>	<b>37</b>
<b>9.a</b>	Recreating the original file using bundle	<b>38</b>	<b>40</b>
<b>9.b</b>	Printing of child and parent process id	<b>41</b>	<b>43</b>
<b>10</b>	Shortest Remaining Time and Round-Robin Scheduling Algorithms	<b>44</b>	<b>45</b>
<b>11</b>	Generating and printing Fibonacci Series	<b>46</b>	<b>47</b>
<b>12</b>	Implementing Banker's algorithm	<b>47</b>	<b>48</b>

**PART A- LEX AND YACC PROGRAMS****Exp.No:1.a****To count number of characters, words, spaces and lines**

**Aim:** Program to count the number of characters, words, spaces and lines in a given input file.

**Algorithm:**

Step1: [Declarations and regular definition:]

Define all C global variable definition and header files to include in the first section. integer character\_count, word\_count, space\_count, line\_count

Step2: [Transition rules]

Declare the transition rules with regular expressions [^

```
\t\n]+ {word_count = word_count + 1
        character_count = character_count + yyleng }
\n      {line_count = line_count + 1}
" "     {space_count = space_count + 1}
\t      {space_count = space_count + 1}
```

Step3: [Auxiliary Procedures]

Define the main() function of C program.

open a file for which action is to be performed and store the file pointer in yyin variable. call yylex() function to perform the analysis.

print the results on the terminal print

```
word_count
print character_count print line_count
print space_count
```

**Program:**

```
% {
int wc = 0, cc = 0, lc = 0, bc=0;
char infile[25];
% }
word [^ \t\n]+
eol \n
%%
{word}      {wc++; cc+=yyleng;}
{eol}      {lc++; cc++; }
[ ]        {bc++; cc++; }
[\t]       {bc+=8; cc++;}
.          {cc++;}
%%
```

```
main()
{
printf(" Read the Input File Name \n ");
scanf("%s",&infile);
yyin=fopen(infile,"r");
yylex();
fclose(yyin);
printf(" No. of Chars: %d\n No. of Words: %d\n No.of Lines: %d\n No. of Blanks:
%d\n",cc,wc,lc,bc); }
```

**Sample Input/Output:**

```
$ cat test1
GCEMM COLLEGE OF ENGINEERING
Department of Computer Science and Engineering
Bangalore
```

```
$ lex prga1a.l
$ cc lex.yy.c -lfl
$ ./a.out test1
no of lines are 3
no of words are 11
no of blanks 10
no of character 82
```

**Exp.No:1.b****To Count the Number of Comment Lines in a given C Program**

**Aim:** Program to count the numbers of comment lines in a given C program. Also eliminate them and copy the resulting Program into separate file.

**Algorithm:**

Step1: [Declarations and regular definition:]

Define all C global variable definition and header files to include in the first section, integer number\_of\_comments

Step2: [Transition rules]

Declare the transition rules with regular expressions

- Rule for multi – line comments  
`("/[*]"[a-zA-Z0-9\n\t]*"*/")` {number\_of\_comments=number\_of\_comments + 1 }
- Rule for single – line comments  
`("/"[n]?a-zA-Z0-9]*)` {number\_of\_comments=number\_of\_comments+ 1 }
- Any thing other than comments to be written in an output file [a-zA-Z0-9(){}.\;#<>]\* {write yytext on yyout }

Step3: [Auxiliary Procedures]

Define the main() function of C program.

Open the input file and store file pointer in yyin variable. Open the output file and store file pointer in yyout variable. Call yylex() function to perform the analysis.

Print the result

print number\_of\_comments

**Program:**

```
% {
    int cc=0;
% }
%x CMNTML CMNTSL
%%
"/[*"                {BEGIN CMNTML;cc++;}
<CMNTML>.          ;
```

```
<CMNTML>\n      ;  
<CMNTML>"*/"  {BEGIN 0;}  
"//"          {BEGIN CMNTSL;cc++;}  
<CMNTSL>.      ;  
<CMNTSL>\n      {BEGIN 0;}  
%%  
main(int argc,char *argv[])
```

```
{
    if(argc!=3)
    {
        printf("usage:%s<src file><dst file>\n",argv[0]);
        return;
    }
    yyin=fopen(argv[1],"r");
    yyout=fopen(argv[2],"w");
    yylex();
    printf("No.of comment lines:%d\n",cc);
}
```

**Sample Input/Output :**

```
$lex sample1.1
$cc lex.yy.c -fl
$./a.out
Write a C program
#include<stdio.h>
int main()
{ int a, b ;
/*float c ;*/
printf(" Hai " );
/*printf("Hello ")* / ;
}
(Note : Press ctrl-d)
Comment=2
$cat output

#include<stdio.h>
int main()
{ int a, b ;
printf("Hai " );
}
```

**Exp.No:2.a****To Recognize a Valid Arithmetic Expression**

**Aim:** Program to recognize a valid arithmetic expression and to recognize the identifiers and operators present. Print them separately.

**Algorithm:**

Step1: [Declarations and regular definition:]

Define all C global variable definition and header files to include in the first section.

```
Integer number_of_plus, number_of_minus, number_of_multiplication integer
number_of_division, number_of_identifiers
integer flag_1, flag_2
```

Step2: [Transition rules]

Declare the transition rules with regular expressions

a) Rule for checking parenthesis

```
[(]      {flag_1 = flag_1 + 1} [)]
        {flag_1 = flag_1 - 1}
```

b) Rule for identifiers

```
[ a-zA-Z0-9]+ {flag_2 = flag_2 + 1; number_of_identifiers = umber_of_identifiers + 1}
```

c) Rule for plus symbol

```
[+]      {flag_2 = flag_2 - 1; number_of_plus = number_of_plus + 1}
```

d) Rule for minus symbol

```
[-]      {flag_2 = flag_2 - 1; number_of_minus = number_of_minus + 1}
```

a) Rule for multiplication symbol

```
[*]      {flag_2 = flag_2 - 1; number_of_multiplication = number_of_multiplication + 1}
```

b)Rule for division symbol

```
[/]      {flag_2 = flag_2 - 1; number_of_division = number_of_division + 1}
```

Step3: [Auxiliary Procedures]

Define the main() function of C program.

Read the input expression from standard input by calling yylex() function Test the validity of the input string by examining the flags

```
flag_1 not equal to 0 or flag_2 not equal to 1 Else print the result
```

```
print number_of_plus print number_of_minus
```

```
print number_of_multiplication print number_of_division
```

```
print number_of_identifiers
```

**Program:**

```

% {
int a[]={0,0,0,0},i,valid=1,opnd=0;
% }
%x OPER
%%
[a-zA-Z0-9]+ { BEGIN OPER; opnd++;}
<OPER>"+" { if(valid) { valid=0;i=0;} else ext();}
<OPER>"-" { if(valid) { valid=0;i=1;} else ext();}
<OPER>"*" { if(valid) { valid=0;i=2;} else ext();}
<OPER>"/" { if(valid) { valid=0;i=3;} else ext();}
<OPER>[a-zA-Z0-9]+ { opnd++; if(valid==0) { valid=1; a[i]++;} else ext();}
<OPER>"\n" { if(valid==0) ext(); else return 0;}
.\n ext();
%%
ext()
{ printf(" Invalid Expression \n"); exit(0); }
main()
{
printf(" Type the arithmetic Expression \n");
yylex();
printf(" Valid Arithmetic Expression \n");
printf(" No. of Operands/Identifiers : %d \n ",opnd);
printf(" No. of Additions : %d \n No. of Subtractions : %d \n",a[0],a[1]);
printf(" No. of Multiplications : %d \n No. of Divisions : %d \n",a[2],a[3]);
}

```

### Sample Input/Output :

```

$ lex prog2a.l
$ cc lex.yy.c -fl
$ ./a.out
enter a valid arithmetic expression
(a+d)/c          Press  ctrl+d
Valid expression
addition(+)=1
subtraction(-)=0
multiplication(*)=0
division(/)=1,id=3

```

**Exp.No:2.b****Sentence is Simple Or Compound**

**Aim:** Program to recognize whether a given sentence is simple or compound.

**Algorithm:**

Step1: [Declarations and regular definition:]

Define all C global variable definition and header files to include in the first section. integer  
flag = 0

Step2: [Transition rules]

Declare the transition rules with regular expressions

a) Rules for testing the sentence

([aA][nN][dD]) { flag = 1 }

("or") { flag = 1 }

("nevertheless") { flag = 1 }

("inspite") { flag = 1 }

Step3: [Auxiliary Procedures]

Define the main() function of C program.

Read the input sentence from the standard input by calling yylex() function Test  
the flag variable to check the sentence

if flag is equal to zero then

print the sentence is simple

else print the sentence is compound

**Program:**

```
% {
/* Program to recognize whether a given sentence is simple or compound.*/
% }
ws [ \n\t]+
%%
{ws}"and"{ws} |
{ws}"AND"{ws} |
```

```
{ws}"or"{ws} |
{ws}"OR"{ws} |
{ws}"but"{ws} |
{ws}"BUT"{ws} |
{ws}"because"{ws} |
{ws}"nevertheless"{ws} {printf("compound sentence\n"); exit(0);}
.;
\n return 0;
%%
main()
{
printf("type the sentence\n");
yylex();
printf("simple sentence\n");
}
```

### **Sample Input/Output :**

#### **Run 1:**

```
$ lex prog2b.l
$ cc lex.yy.c -lfl
$ ./a.out
```

enter the sentence

Hi and Hello

sentence is compound

#### **Run 2:**

```
    $ ./a.out
enter the sentence
```

Hi Hello

Sentence is simple

**Exp.No:3****Count the Number of identifiers**

**Aim:** Program to recognize and count the number of identifiers in a given input file.

**Algorithm:**

Step1: [Declarations and regular definition:]

Define all C global variable definition and header files to include in the first section. integer count

Step2: [Transition rules]

Declare the transition rules with regular expressions Rules for identifiers in a source C program

“int” | “float” | “double” |

“char” { read a character by calling input() function and store in a variable say *ch*

repeat for ever

test *ch* if it is “;” the count = count + 1 test *ch* if it is “\n”

then break the loop read the next character and store in

*ch*}

Step3: [Auxiliary Procedures]

Define the main() function of C program.

Open the input file and store file pointer in *yyin* variable. Call *yylex()* function to perform the analysis.

Print the result

print count

**Program:**

```
% {
/* 3.1 Lex Program to Recognize and count the Identifiers in a given input file ( C Program) */
int idc=0;
% }
WS [ \t\n]*
ID [_a-zA-Z][_a-zA-Z0-9]*
DECLN "int"|"float"|"char"|"short"|"long"|"unsigned"
%x DEFN
%%
{DECLN} {BEGIN DEFN;}
<DEFN>{WS}{ID}{WS}\,    idc++;
<DEFN>{WS}{ID}{WS}\;    idc++;
<*>\n ;
<*>. ;
%%
main(int argc, char **argv)
{
```

```
if(argc==2)
{
yyin=fopen(argv[1],"r");
yylex();
printf(" Total No. of Identifiers : %d\n",idc);
}
else
printf(" Usage : %s <file> \n\n",argv[0]);
}
```

**Sample Input/Output:**

```
$cat > input6
int
float
78f
90gh
a
d
are case
default
printf
scanf
$lex 3.1
$cc lex.yy.c -lfl
$./a.out
```

```
Enter the file name: input6
int is a keyword
78f is not an identifier
90gh is not an identifier
a is an identifier
d is an identifier
are is an identifier
case is a keyword
default is a keyword
printf is a keyword
scanf is a keyword
total identifiers are: 3
```

**Exp.No:4.a****To Recognize a Valid Arithmetic Expression**

**Aim:** Program to recognize a valid arithmetic expression that uses operators +, -, \* and /.

**Algorithm: Lex**

- Step1: [Declarations and regular definition:]  
         Define head files to include in the first section.
- Step2: [Transition rules]  
         Tokens generated are used in yacc file [a-zA-Z] Alphabets are returned.
- Step3 : 0-9 one or more combination of Integers.

**Yacc**

- Step1: Define head file to include in the first section.
- step2: Accept token generated in lex part as input.
- step3:Specify the order of procedure.
- Step4:Transition rules  
         Define the rules with end points.
- Step5:Parse input string form standard input by calling yyparse() in main function.  
         Print the result of any of the rules defined matches. Arithmetic expression is valid.
- Step6:If none of the rules defined matches. Print Arithmetic expression is invalid.

**Program:****Lex Part:**

```
% {
#include "y.tab.h"
% }
%%
[0-9]+(\.[0-9]+)?      { return NUM; }
[a-zA-Z][_a-zA-Z0-9]* { return ID; }
[\t]                  ;
\n                    return 0;
.                      return yytext[0];
%%
```

**Yacc Part:**

```
% {
#include<stdio.h>
% }
%token NUM ID
%left '+' '-'
%left '*' '/'
%%
e : e '+' e
  | e '-' e
  | e '*' e
  | e '/' e
  | '('e')
  | NUM
  | ID      ;
%%
main()
{
printf(" Type the Expression & Press Enter key\n");
yyparse();
printf(" Valid Expression \n");
}
yyerror()
{
printf(" Invalid Expression!!!\n");
exit(0);
}
```

**Sample Input/Output:**

```
$lex 4a.l
$yacc -d 4a.y
$cc lex.yy.c y.tab.c -fl
$./a.out
```

```
Enter the expression
a+b
Expression is valid
```

```
./a.out
```

```
Enter the expression
(a+b) * (c+d) + (m+
Expression is invalid
```

**Exp.No:4.b****To Recognize a Valid Variable**

**Aim:** Program to recognize a valid variable, which starts with a letter, followed by any number of letters or digits.

**Algorithm: Lex:**

- step1. Define head file to include in the first section.
- step2. Accept token generated in lex part as input.
- step3. Specify the order of procedure.
- Step4. Define header file to include in the first section.
- Step5. Transition rules
  - a. [a-z] letters are returned
  - b. [0-9] digits are returned

**Yacc:**

- step1. Step1: Define head file to include in the first section.
- step2. Accept token generated in lex part as input.
- Step3. Transition rules
  - Define the rules with end points.
- Step4. Parse input string from standard input by calling yyparse(); in main() function.
  - Print the result of any of the rules defined matches Valid variable
- Step5. If none of the rules defined matches. Print Invalid variable

**Program:****Lex part**

```
% {
#include "y.tab.h"
% }
%%
[a-z] return L;
[0-9] return D;

%%
```

**Yacc part**

```
% {
% }
%token L D
%%
var : L E { printf(" Valid Variable \n"); return 0; }
E : E L ;
| E D ;
| ;
%%
main()
```

```
{ printf(" Type the Variable \n"); yyparse();  
  } yyerror()  
  { printf(" Invalid variable !!!\n"); exit(0); }
```

**Sample Input/Output:**

\$lex 4b.l

\$yacc -d 4b.y

\$cc lex.yy.c y.tab.c -lfl

\$/a.out

Sum6

The string is a valid variable

\$/a.out

4Sum

The string is not a valid variable

**Exp.No:5.a****Evaluate an Arithmetic Expression**

**Aim:** Program to evaluate an arithmetic expression involving operators +, -, and /.

**Algorithm: Lex**

Step1: Declaration section

Define header file and c global variable definition in the first section.

Step2: Transition rules

[0-9] one or more combination of integers.

**Yacc**

Step1: Declaration section.

Define header file to include in the first section.

Step2: Accept the token generated in lex part as input.

Step3: Specify the order of procedure.

Step4: Transition rules.

a. Define the rules with end point

b. Parse input string from standard input by calling yyparse(); by in main function.

Step5: Print the result if any of the rules defined matches.

Step6: If none of the rules defined matches. Print Invalid expression.

**Program:****Lex Part**

```
%{
#include<stdlib.h>
#include "y.tab.h"
extern int yylval;
%}
%%
[0-9]+      { yylval=atoi(yytext);    return NUM;  }
[\t]        ;
\n          return 0;
.           return yytext[0];
%%
```

**Yacc Part**

```
%{
#include <stdio.h>
%}
%token NUM
%left '+' '-'
%left '*' '/'
%%
```

```
expr : e { printf(" Result : %d\n", $1); return 0; };
e : e '+' e { $$=$1+$3; }
| e '-' e { $$=$1-$3; }
| e '*' e { $$=$1*$3; }
| e '/' e { $$=$1/$3; }
| '('e' { $$=$2; }
| NUM    { $$=$1; };
%%
main()
{
printf(" Type the Expression & Press Enter key\n"); yyparse();
printf(" Valid Expression \n");
}
yyerror()
{ printf(" Invalid Expression!!!!\n"); exit(0); }
```

### **Sample Input/Output:**

\$lex 5a.l

\$yacc -d 5a.y

\$cc lex.yy.c y.tab.c -lfl

./a.out

Enter the expression in terms of integers

7\*2

14

Success

./a.out

Enter the expression in terms of integers

9-2

7

Success

**Exp.No:5.b****To Recognize Strings “aaab abbb ab and a”**

**Aim:** Program to recognize strings „aaab“, „abbb“, „ab“ and „a“ using the grammar ( $a^n b^n$ ,  $n \geq 0$ ).

**Algorithm: Lex**

Step1: Define head file to include in the first section.

Step2: Transition rules.

Example

I. a A is returned

II. b B is returned

**Yacc :**

Step1: Include global c declaration and assign it to one.

Step2: Accept token generated in lex part as input.

Step3: Define header file to include in the first section.

Step4: Transition rules.

a. Define the rules with end point

b. Parse input string from standard input by calling `yyparse()`; by in main function.

Step5: Print the result Valid string if any of the rules defined matches.

Step6: If none of the rules defined matches print Invalid string.

**Program:****Lex Part**

```
% {
#include "y.tab.h"
% }
%%
a      return A;
b      return B;
.      return yytext[0];
\n     return yytext[0];
%%
```

**Yacc part**

```
%token A B
%%
str: s '\n' { return 0;} s : A s B ;
| ;
%%
main()
{
printf(" Type the String ?\n"); if(!yyparse())
```

```
printf(" Valid String\n "); } int yyerror()
{
printf(" Invalid String.\n"); exit(0);
}
```

**Sample Input/Output:**

```
$lex sample5.l
$yacc -d sample5.y
$cc lex.yy.c y.tab.c -lfl
$./a.out
enter the string
aabb
(Note: press ctrl-d)
```

```
valid
$./a.out
enter the string
aab
syntax error
```

## Exp.No:6

### To Recognize the grammar ( $a^n b$ , $n \geq 10$ )

**Aim:** Program to recognize the grammar ( $a^n b$ ,  $n \geq 10$ )

#### Algorithm:

##### Lex

Step1: Define head file to include in the first section.

Step2: Transition rules.

Example

a A is  
returned b  
B is  
returned

##### Yacc

Step1: Include global c declaration and assign it to one.

Step2: Accept token generated in lex part as input.

Step3: Define header file to include in the first section.

Step4: Transition rules.

c. Define the rules with end point

d. Parse input string from standard input by calling `yyparse()`; by in main function.

Step5: Print the result Valid string if any of the rules result defined matches.

Step6: If none of the rules defined matches print Invalid string.

#### Program:

##### Lex Part

```
% {  
#include "y.tab.h"  
% }  
%%  
  
a      return A;  
b      return B;  
.      return yytext[0];
```

```
\n    return yytext[0];
%%
```

### **Yacc part**

```
% {
% }
%token A B
%%
str: s '\n' { return 0;}
s : x B ;
x : A A A A A A A A A T ;
T: T A
| ;
%%
main()
{
printf(" Type the String ? \n");
if(!yyparse())
    printf(" Valid String\n ");
}
int yyerror()
{
printf(" Invalid String.\n");
exit(0);
}
```

### **Sample Input/Output:**

```
$lex 6.1
$yacc -d 6.y
$cc lex.yy.c y.tab.c -lfl
$./a.out
enter the string
aaaaaaaaaaaab
valid
$./a.out
enter the string
aab

error
```

**PART – B****UNIX PROGRAMMING****Exp.No:7.a****Reverse the number of arguments**

**Aim:** Non-recursive shell script that accepts any number of arguments and prints them in the Reverse order. (For example, if the script is named arguments, then executing arguments A B C should produce C B A on the standard output).

**Algorithm:**

```
#!/bin/sh
# Program to reverse the input arguments
if test $# -eq 0
then
    echo "Not enough arguments to be displayed"
else
    echo "number of arguments are:$#"
    echo "the arguments passed is $*"
    echo "the reverse of it is"
    c=$#
    while [ $c -ne 0 ]
    do
        eval echo \$$c
        c=`expr $c - 1`
    done
    echo "End of the program"
fi
exit 0
```

**Program:**

```
#!/bin/sh
#program to reverse command line arguments
if test $# -eq 0
then
    echo "Not enough arguments to be displayed"
else
    for actstr in $*
    do
        rev=`echo $actstr $rev`
    done
    echo $rev
fi
exit 0
```

**Sample Input/Output:**

```
[root@localhost]# sh 7a1.sh A B C
number of arguments are:3
the arguments passed is A B C
the reverse of it is
C
B
A
End of the program
```

## Exp.No:7.b

### Implementing child process

**Aim:** C program that creates a child process to read commands from the standard input and execute them (a minimal implementation of a shell -like program). You can assume that no arguments will be passed to the commands to be executed.

**Program:**

```
#include<stdio.h>
#include<unistd.h>
int main()
{
    int x,ch;
    char cmd[20];
    int pid=fork();
    if(pid==0) /* child process execution */
    {
        printf("\n Child process");
        do
        {
            printf("\n Enter the command:");
            scanf("%s",&cmd);
            system(cmd);
            printf("\n Enter 1 to continue or 0 to exit:");
```

```
        scanf("%d",&ch);
    }while(ch!=0);
}
else/* parent execution*/
    wait(); /* Block the parent until the completion of the child*/
}
```

**Sample Input/Output:**

```
[root@localhost]# cc 7b.c  
[root@localhost]# ./a.out
```

Child process

Enter the command:date

```
Thu Jun27 13:17:32 IST 2013
```

Enter 1 to continue or 0 to exit:1

Enter the command:ls

```
1a.sh 2a.sh 3a.sh 4a.sh 5a.sh 6a.sh 7a.sh 1b.c 2b.c 3b.c 4b.c 5b.c 6b.c 7b.c a.out cc
```

Enter 1 to continue or 0 to exit:1

Enter the command:who

```
root  :0      Sep 10 12:54
```

```
root  pts/1   Sep 10 12:55 (:0.0)
```

Enter 1 to continue or 0 to exit:1

Enter the command:ps

```
PID TTY      TIME CMD
```

```
3620 pts/1  00:00:00 bash
```

```
3705 pts/1  00:00:00 a.out
```

```
3706 pts/1  00:00:00 a.out
```

```
3711 pts/1  00:00:00 ps
```

Enter 1 to continue or 0 to exit:1

Enter the command:pwd

```
/root/unix
```

Enter 1 to continue or 0 to exit:0

```
[root@localhost]#
```

**Exp.No:8.a****Checking the file permissions**

**Aim:** Shell script that accepts two filenames as arguments, checks if the permissions for these files are identical and if the permissions are identical, outputs the common permissions, otherwise outputs each filename followed by its permissions.

**Program:**

```
#!/bin/sh
#program to check file permission
if [ $# -ne 2 ]
then
    echo —execution is sh 2a.sh first_file second_file
    exit 2
fi
#permission of first file
p1=`ls -l $1 | cut -d " " -f 1`
#permission of second file
p2=`ls -l $2 | cut -d " " -f 1`
if test $p1 = $p2
then
    echo "The permissions are same"
else
    echo $p1
    echo "The permissions are different"
    echo "Filename: $1 permission: $p1 "
    echo "Filename: $2 permission: $p2"
fi
exit 0
```

**Sample Input/Output:**

```
[root@localhost]# sh 8a.sh aaa tem
The permissions are different
Filename: aaa  permission: drwxr-xr-x
Filename: tem  permission: -rw-r--r--
```

```
[root@localhost]# sh 8a.sh tem temp
The permissions are same
-rw-r--r--
```

## Exp.No:8.b

**To demonstrate how the hole in file is handled.**

**Aim:** C program to create a file with 16 bytes of arbitrary data from the beginning and another 16 bytes of arbitrary data from an offset of 48. Display the file contents to demonstrate how the hole in file is handled.

### Program:

```
#include<stdio.h>
#include<stdlib.h>
#include<fcntl.h>
#include<sys/stat.h>
#include<sys/types.h>
int main()
{
    char buf1 []="abcdefghijklmnop";
    char buf2 []="ABCDEFGHIJKLMNOP";
    int fd;
    // creating file new.dat
    fd=open("new.dat",O_WRONLY|O_CREAT,0777);
    //writing 16 bytes of data
    write(fd,&buf1,16);
    // offset of 48
    lseek(fd,48,SEEK_SET);
    // writing 16 bytes of data
    write(fd,&buf2,16);
    close(fd);
    printf("contents of file are\n");
    system("od -c new.dat");
    exit(0);
}
```

### Sample Input/Output:

```
[root@localhost]# cc 8b.c
[root@localhost]# ./a.out
0000000 a b c d e f g h i j k l m n o p
0000020 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0
*
0000060 A B C D E F G H I J K L M N O P
0000100
```

**Exp.No:9.a****Implementing Bundle Script**

**Aim:** Shell script that accepts file names specified as arguments and creates a shell script that contains this file as well as the code to recreate these files. Thus if the script generated by your script is executed, it would recreate the original files(This is same as the “bundle” script described by Brain W.Kernighan and Rob Pike in “The Unix Programming Environment”, Prentice – hall India).

**Program:**

```
#!/bin/sh
#program to create a bundle script
if [ $# -eq 0 ]
then
    echo "no arguments passed"
    exit 1
fi
echo " ">cr.sh
for i in $*
do
    echo "echo "code to recreate the file $i"">>cr.sh
    echo "cat $i<<endf">>cr.sh
    cat $i>> cr.sh
    echo "endf">>cr.sh
    echo "echo "creation done"">>cr.sh
done
exit 0
```

**Sample Input/Output:**

```
[root@localhost]# cat >f1<<endf
CITECH
endf
[root@localhost]# cat >f2<<endf
BANGALORE
endf
[root@localhost]# sh 9a.sh f1 f2
[root@localhost]# vi cs.sh
echo code to recreate the file f1
cat f1<<endf
CITECH
endf
echo creation done
echo code to recreate the file f2
cat f2<<endf
BANGALORE
endf
echo creation done
```

## Exp.No:9.b

### Impementing parent and child process PID's

**Aim:** C program to do the following: Using fork() create a child parent and then exits. the parent process waits for its child to finish(by executing the wait()) and prints its own process-id and the id of its child process and then exits.

**Algorithm:**

Step1: Start the execution

Step2: Create process using fork and assign it to a variable

Step3: Check for the condition pid is equal to 0

Step4: If it is true print the pid of the child process with its parent &terminate the child process.

Step5: If it is not a parent process has to wait until the child terminate & then print its pid & its child pid.

Step6: Stop the execution

**Program:**

```
#include<sys/types.h>
#include<stdio.h>
int main()
{ pid_t pid;
  if (pid=fork()) < 0)
    printf("\nFork error");
  if(pid==0)
  {
    printf("\nThis is Child Process");
    printf("\nChild PID: %d",getpid());
    printf("\nParent PID:%d\n",getppid());
    exit(0);
  }
  else
  {
    wait();
    printf("\nThis is Parent Process");
    printf("\nParent PID:%d",getpid());
    printf("\nChild PID:%d",pid);
    exit(0);
  }
  return 0;
}
```

**Sample Input/Output:**

a)

\$cc 9b.c  
\$./a.out

This is child process  
Child PID:3122  
Parent PID:3121

This is Parent Process  
Parent PID: 3121  
Child PID : 3122

b)

\$cc 9b.c  
\$./a.out

Child process  
My PID=3902  
My parent PID=3901

Parent process  
my PID=3901  
My Child PID=3902

## Operating System Programs

### Exp.No:10

#### Implementing SRTF and Round Robin Scheduling Algorithms

**Aim:** Design, develop and execute a program in C/C++ to simulate the working of Shortest Remaining time and Round Robin Scheduling algorithms. Experiment with different Quantum sizes for the Round Robin algorithm. In all cases, determine the average turn Around time. Input can be read from keyboard or from a file.

#### Algorithm:

- Step1: Start the process
- Step2: Get the number of elements to be inserted
- Step3: Get the value for burst time for individual processes
- Step4: Get the value for time quantum
- Step5: Make the CPU scheduler go around the ready queue allocating CPU to each process for the time interval specified
- Step6: Make the CPU scheduler pick the first process and set time to interrupt after quantum. And after it's expiry dispatch the process
- Step7: If the process has burst time less than the time quantum then the process is released by the CPU
- Step8: If the process has burst time greater than time quantum then it is interrupted by the OS and the process is put to the tail of ready queue and the schedule selects next process from head of the queue
- Step9: Calculate the total and average waiting time and turn around time
- Step10: Display the results
- Step11: Stop the process

#### Program:

```
#include<stdio.h>
struct proc
{
    int id;
    int arrival;
    int burst;
    int rem;
    int wait;
    int finish;
    int turnaround;
    float ratio;
}process[10]; //structure to hold the process information
struct proc temp;
```

```

int no;

int chkprocess(int);
int nextprocess();
void roundrobin(int, int, int[], int[]);
void srtf(int);

main()
{
    int n,tq,choice;
    int bt[10],st[10],i,j,k;
    for( ; )
    {
        printf("Enter the choice \n");
        printf(" 1. Round Robin\n 2. SRT\n 3. Exit \n");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1:
                printf("Round Robin scheduling algorithm\n");
                printf("Enter number of processes:\n");
                scanf("%d",&n);
                printf("Enter burst time for sequences:");
                for(i=0;i<n;i++)
                {
                    scanf("%d",&bt[i]);
                    st[i]=bt[i]; //service time
                }
                printf("Enter time quantum:");
                scanf("%d",&tq);
                roundrobin(n,tq,st,bt);
                break;

            case 2:
                printf("\n \n ---SHORTEST REMAINING TIME NEXT---\n \n ");
                printf("\n \n Enter the number of processes: ");
                scanf("%d", &n);
                srtf(n);
                break;

            case 3: exit(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,j,k;
    float awt=0.0,atat=0.0;
    while(1)
    {

```

```

    for(i=0,count=0;i<n;i++)
    {
        temp1=tq;
        if(st[i]==0) // when service time of a process equals zero then
                    //count value is incremented
        {
            count++;
            continue;
        }
        if(st[i]>tq) // when service time of a process greater than time
                    //quantum then time
            st[i]=st[i]-tq; //quantum value subtracted from service time
        else
            if(st[i]>=0)
            {
                temp1=st[i]; // temp1 stores the service time of a process
                st[i]=0; // making service time equals 0
            }
            sq=sq+temp1; // utilizing temp1 value to calculate turnaround time
            tat[i]=sq; // turn around time
        } //end of for
    if(n==count) // it indicates all processes have completed their task because the count value
    break; // incremented when service time equals 0
} //end of while

for(i=0;i<n;i++) // to calculate the wait time and turnaround time of each process
{
    wt[i]=tat[i]-bt[i]; // waiting time calculated from the turnaround time - burst time
    swt=swt+wt[i]; // summation of wait time
    stat=stat+tat[i]; // summation of turnaround time
}
awt=(float)swt/n; // average wait time
atat=(float)stat/n; // average turnaround time
printf("Process_no Burst time Wait time Turn around time\n");
for(i=0;i<n;i++)
    printf("%d\t%d\t%d\t%d\n",i+1,bt[i],wt[i],tat[i]);
printf("Avg wait time is %f\n Avg turn around time is %f\n",awt,atat);
} // end of Round Robin

int chkprocess(int s) // function to check process remaining time is zero or not
{
    int i;
    for(i = 1; i <= s; i++)
    {
        if(process[i].rem != 0)
            return 1;
    }
    return 0;
} // end of chkprocess

int nextprocess() // function to identify the next process to be executed

```

```

{
    int min, l, i;
    min = 32000; //any limit assumed
    for(i = 1; i <= no; i++)
    {
        if( process[i].rem!=0 && process[i].rem < min)
        {
            min = process[i].rem;
            l = i;
        }
    }
    return l;
} // end of nextprocess

void srtf(int n)
{
    int i,j,k,time=0;
    float tavg,wavg;
    for(i = 1; i <= n; i++)
    {
        process[i].id = i;
        printf("\n\nEnter the arrival time for process %d: ", i);
        scanf("%d", &(process[i].arrival));
        printf("Enter the burst time for process %d: ", i);
        scanf("%d", &(process[i].burst));
        process[i].rem = process[i].burst;
    }
    for(i = 1; i <= n; i++)
    {
        for(j = i + 1; j <= n; j++)
        {
            if(process[i].arrival > process[j].arrival) // sort arrival time of a process
            {
                temp = process[i];
                process[i] = process[j];
                process[j] = temp;
            }
        }
    }

    no = 0;
    j = 1;
    while(chkprocess(n) == 1)
    {
        if(process[no + 1].arrival == time)
        {
            no++;
            if(process[j].rem==0)
                process[j].finish=time;
            j = nextprocess();
        }
        if(process[j].rem != 0) // to calculate the waiting time of a process

```

```

        {
            process[j].rem--;
            for(i = 1; i <= no; i++)
            {
                if(i != j && process[i].rem != 0)
                    process[i].wait++;
            }
        }
    else
    {
        process[j].finish = time;
        j=nextprocess();
        time--;
        k=j;
    }

    time++;
}
process[k].finish = time;

printf("\n\n\t\t\t---SHORTEST REMAINING TIME FIRST---");
printf("\n\n Process Arrival Burst Waiting Finishing turnaround Tr/Tb \n");
printf("%5s %9s %7s %10s %8s %9s\n\n", "id", "time", "time", "time", "time", "time");
for(i = 1; i <= n; i++)
{
    process[i].turnaround = process[i].wait + process[i].burst; // calc of turnaround
    process[i].ratio = (float)process[i].turnaround / (float)process[i].burst;

    printf("%5d %8d %7d %8d %10d %9d %10.1f ", process[i].id, process[i].arrival,
    process[i].burst, process[i].wait, process[i].finish, process[i].turnaround,
    process[i].ratio);
    tavg=tavg+ process[i].turnaround; //summation of turnaround time
    wavg=wavg+process[i].wait; // summation of waiting time
    printf("\n\n");
}

tavg=tavg/n; // average turnaround time
wavg=wavg/n; // average wait time
printf("tavg=%f\t wavg=%f\n",tavg,wavg);
} // end of srtf

```

**Sample Input/Output:**

Enter the choice

1) Round Robin 2) SRT

3) Exit

1

Round Robin scheduling algorithm

\*\*\*\*\*

Enter number of processes:3

Enter burst time for sequences:24

3

3

Enter time quantum:4

Process_no	Burst time	Wait time	Turnaround time
1	24	6	30
2	3	4	7
3	3	7	10

Avg wait time is 5.666667

Avg turnaround time is 15.666667

Enter the choice

1) Round Robin 2) SRT

3) Exit

2

---SHORTEST REMAINING TIME NEXT---

Enter the number of processes: 4

Enter the arrival time for process 1: 0

Enter the burst time for process 1: 8

Enter the arrival time for process 2: 1

Enter the burst time for process 2: 4

Enter the arrival time for process 3: 2

Enter the burst time for process 3: 9

Enter the arrival time for process 4: 3

Enter the burst time for process 4: 5

1	24	6	30
---	----	---	----

2	3	4	7
---	---	---	---

3	3	7	10
---	---	---	----

---SHORTEST REMAINING TIME FIRST---

Enter the number of processes: 4

Enter the arrival time for process 1: 0

Enter the burst time for process 1: 8

Enter the arrival time for process 2: 1

Enter the burst time for process 2: 4

Enter the arrival time for process 3: 2

Enter the burst time for process 3: 9

Enter the arrival time for process 4: 3

Enter the burst time for process 4: 5

---SHORTEST REMAINING TIME NEXT---

Process Arrival Burst Waiting Finishing turnaround Tr/Tb

id	time	time	time	time	time	time
1	0	8	9	17	17	2.1
2	1	4	0	5	4	1.0
3	2	9	15	26	24	2.7
4	3	5	2	10	7	1.4

tavg=13.000000

wavg=6.500000

## Exp.No:11

### Implementing Multi-Threading using OpenMp

**Aim:** Design develop and run a multi-threaded program to generate and print Fibonacci series. One thread has to generate the numbers up to the specified limit and Another thread has to print them. Ensure proper synchronization.

**Program:**

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

int MAX;

int Fibonacci(int n)
{
    int x, y;
    if (n < 2)
        return n;
    else
    {
        x = Fibonacci(n - 1);
        y = Fibonacci(n - 2);
        return (x + y);
    }
}

int FibonacciTask(int n)
{
    int x, y;
    if (n < 2)
        return n;
    else
    {
        x = Fibonacci(n - 1);
        y = Fibonacci(n - 2);
        return (x + y);
    }
}

/* random number generation upto 24 */
int random_num()
{
    int temp;
    temp = rand();
    temp = temp%24;
    MAX = temp;
    return(MAX);
}

int main(int argc, char * argv[])
{
```

```

int FibNumber[25] = {0};
int j, temp,tmp,id,i = 0;
int n, tid, nthreads;

printf("Please Enter the number Range :");
scanf("%d",&n);
printf("\n");
omp_set_num_threads(2);

//Parallel region
# pragma omp parallel
{
    printf("The number of threads are %d\n",omp_get_num_threads());
    # pragma omp for private (tid, tmp, FibNumber)
    for(j = 1; j<=n; j++)
    {
        tmp = random_num();
        /* Get thread number */
        /* tid = omp_get_thread_num();
        printf("The number of threads are %d\n",omp_get_num_threads());
           printf("The thread id is = %d\n", tid); */
        /* The critical section here will enable, not more then one
           thread to execute in this section (synchronization) */
        # pragma omp critical
        {
            /* Get thread number */
            /* tid = omp_get_thread_num();
            printf("***** inside critical section
               *****\n");
            printf("The thread id is = %d\n", tid); */
            for(i = 1; i <= tmp; i++)
                FibNumber[i] = FibonacciTask(i);
            printf("The number value is %d:",tmp);
            for(i = 1; i <= tmp; i++)
                printf("%d \t", FibNumber[i]);
            printf("\n\n");
        }
    }
}
}

```

**Sample Input/Output:**

```
$cc fib.c -fopenmp
```

```
$ ./a.out
```

```
Enter the Limit:5
```

```
There are 2 threads
```

```
Thread 0 generating numbers
```

```
Thread 1 printing numbers
```

```
0 1 1 2 3
```

**Exp.No:12****Implement Banker's Algorithm**

**Aim:** Design, develop and run a program to implement the Banker's Algorithm. Demonstrate its Working with different data values.

**Algorithm:** To check if a state is safe or not:

- i. Look for a row R whose count unmet resources needs are all smaller than A. if no such row exists; the system is deadlocked since no process can run to completion.
  - Assume the process of the row chosen requests all the resources it needs & finishes. Mark the process as terminated & add its resources to A vector.
- ii. Repeat steps i & ii, until either all processes are marked terminated.

If several processes are eligible to be chosen in step1, it does not matter which one is selected.

**Program:**

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
void main()
{
    int max[10][10],need[10][10],alloc[10][10],avail[10],completed[10];
    int p,r,i,j,process,count=0;
    printf("Enter the number of processes");
    scanf("%d",&p);
    for(i=0;i<p;i++)
        completed[i]=0;
    printf("\nEnter the number of resources:");
    scanf("%d",&r);
    printf("\nEnter the max matrix");
    for(i=0;i<p;i++)
        for(j=0;j<r;j++)
            scanf("%d",&max[i][j]);
    printf("Enter the allocation matrix:");
    for(i=0;i<p;i++)
        for(j=0;j<r;j++)
            scanf("%d",&alloc[i][j]);
    printf("Enter the available resource:");
    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
    {
        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)
{
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("The system is in safe state\n");
else
printf("The system is in unsafe state\n");
getch();
}

```

### **Sample Input/Output:**

Enter the maximum matrix :

3 2 2 6 1 3 3 1 4 4 2 2

Enter the allocation matrix:

1 0 0 5 1 1 2 1 1 0 0 2

Enter the available resource:

1 1 2

The system is in safe state

\$/a.out

Enter the no. of Processes: 4

Enter the no. of Resources: 2

Enter the maximum matrix :

9 5 2 6 2 2 5 0

Enter the allocation matrix:

7 2 1 3 1 1 3 0

Enter the available resource:

2 1

The system is in unsafe state

