SRI KRISHNA COLLEGE OF ENGINEERING AND TECHNOLOGY
(An Autonomous Institution. Affiliated to Anna University, Chennai)
Kuniamuthur, Coimbatore - 641 008

DEPARTMENT OF INFORMATION TECHNOLOGY

II B.Tech- Information Technology

22CS403 - OPERATING SYSTEMS LAB

PRACTICAL RECORD

Submitted by

Name : .....

Reg.No : .....

SRI KRISHNA COLLEGE OF ENGINEERING AND TECHNOLOGY
(An Autonomous Institution. Affiliated to Anna University, Chennai)
Kuniamuthur, Coimbatore - 641 008

DEPARTMENT OF INFORMATION TECHNOLOGY

22CS403 - OPERATING SYSTEMS LAB

PRACTICAL RECORD

Name : …… **Reg.no :……..……**

Class : II BTECH IT B **Semester : IV**

BONAFIDE CERTIFICATE

Certified bonafide record **of work done by Mr. /Ms …….**

Reg No. ……..…. during the academic year 2023-2024

Submitted for the end semester practical examination held on ……....

Staff-In Charge **HOD**

INTERNAL EXAMINER **EXTERNAL EXAMINER**

INDEX

SRI KRISHNA COLLEGE OF ENGINEERING AND TECHNOLOGY
(An Autonomous Institution. Affiliated to Anna University, Chennai)
Kuniamuthur, Coimbatore - 641 008

Department of IT

Rubrics for Evaluating Laboratory

Subject Code : 22CS403

Lab Name : Operating Systems Lab

Method: Lab Reports and Observation of Faculty Incharge

Outcomes Assessed:

a) Graduates will demonstrate knowledge of mathematical, scientific and multidisciplinary approach for problem solving.

b) Graduates will be able to apply their knowledge in various programming skills to create solutions for product based and application based software.

c) Graduates will possess the ability to create real time solutions for different projects by using modern tools prevailing in the current trends.

e) Graduates attain advanced knowledge in the stream of Information Technology and basic knowledge in Electronics and Communication Engineering to develop and maintain the simple and complex information systems.

SRI KRISHNA COLLEGE OF ENGINEERING AND TECHNOLOGY
(An Autonomous Institution. Affiliated to Anna University, Chennai)
Kuniamuthur, Coimbatore - 641 008

Department of IT

Register Number :

Name of the Student :

Name of the lab : 22CS403 Operating Systems Lab

Components	Exp No and Date												Average Score
	Ex1	Ex2	Ex3	Ex4	Ex5	Ex6	Ex7	Ex8	Ex9	Ex10	Ex11	Ex12	
Aim & Algorithm 20 Marks													
Coding 30 Marks													
Compilation & Debugging 30 Marks													
Execution & Results 10 Marks													
Documentation & Viva 10 Marks													
Total													

Staff In-charge

PROGRAMME OUTCOMES

a) Graduates will demonstrate knowledge of mathematical, scientific and multidisciplinary approach for problem solving. ***(Criteria to be used for assessment Aim, Algorithm, Flowchart (Optional) and Description with sample Test cases, Coding, Compilation and Debugging)***

b) Graduates will be able to apply their knowledge in various programming skills to create solutions for product based and application based software. ***(Criteria to be used for assessment Coding, Compilation and Debugging)***

c) Graduates will possess the ability to create real time solutions for different projects by using modern tools prevailing in the current trends. ***(Criteria to be used for assessment Aim, Algorithm, Flowchart (Optional) and Description with sample Test cases, Coding, Compilation and Debugging, Execution and Results (Inclusion of Generalization like Subroutines, Modules)***

e) Graduates attain advanced knowledge in the stream of Information Technology and basic knowledge in Electronics and Communication Engineering to develop and maintain the simple and complex information systems. ***(Criteria to be used for assessment Aim, Algorithm, Flowchart (Optional) and Description with sample Test cases, Coding, Compilation and Debugging, Execution and Results (Inclusion of Generalization like Subroutines, Modules***

Staff In-charge

Name:
Reg. No:

Ex no: 01

Date: 03.05.2024

Basic Linux Commands

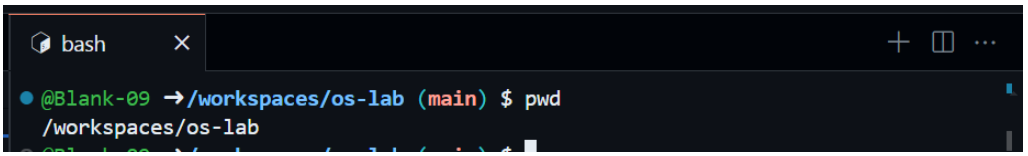
AIM:

To study the basic commands in Linux.

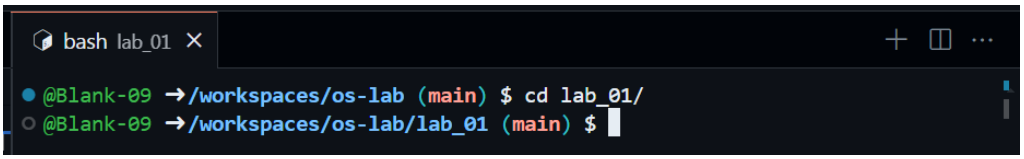
COMMANDS:

1. **Command:** ls
   Description: Lists the files and directories in the current directory.
   Syntax: `ls [options] [directory]`

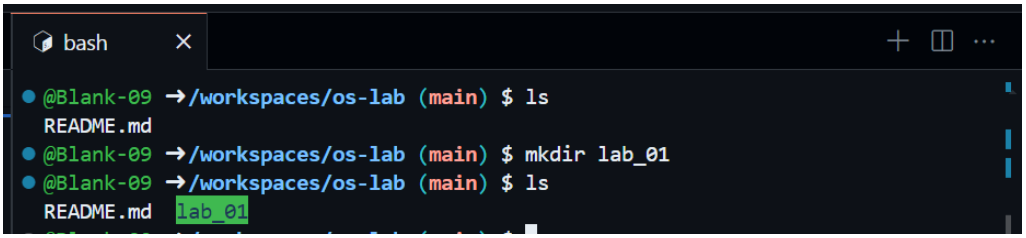
2. **Command:** pwd
   Description: Prints the current working directory.
   Syntax: pwd



3. **Command:** cd
   Description: Changes the current directory.
   Syntax: `cd [directory]`



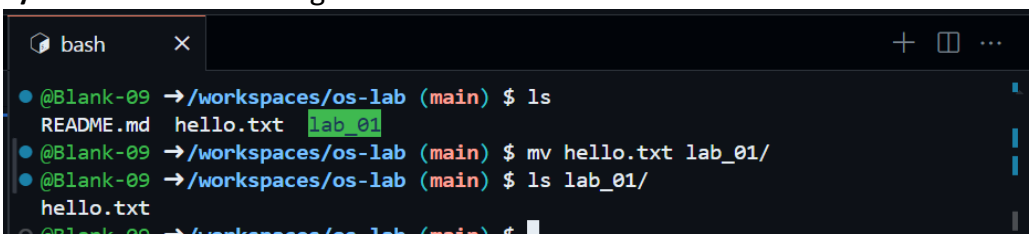
4. **Command:** mkdir
   Description: Creates a new directory.
   Syntax: `mkdir directory`



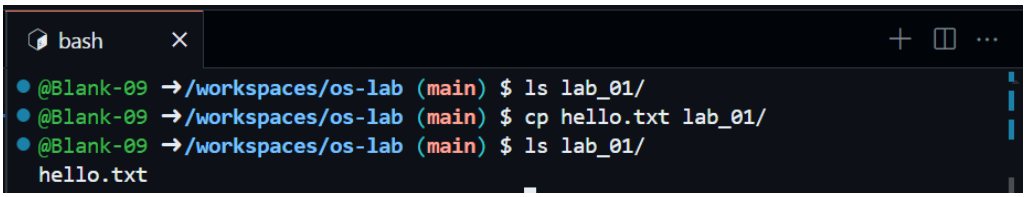
5. **Command:** touch
   Description: Creates an empty file or updates the timestamp of an existing file.
   Syntax: `touch filename`

Name:
Reg. No:

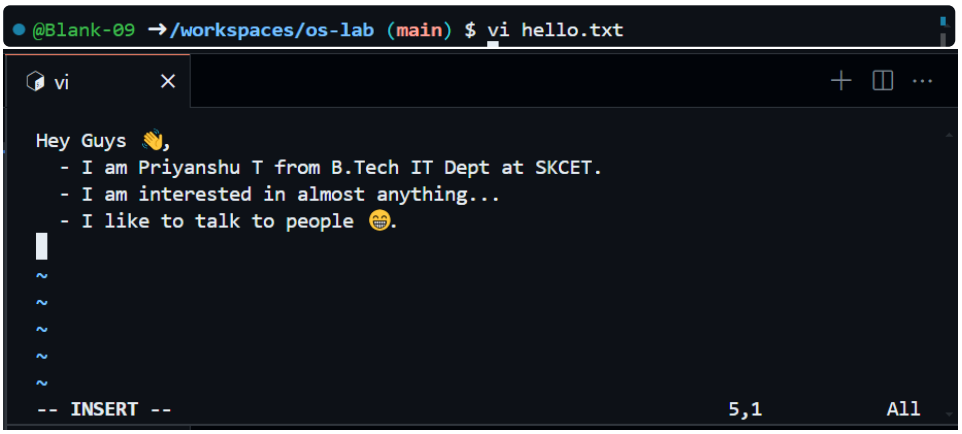
6. **Command:** mv
   Description: Moves or renames files or directories.
   Syntax: `mv source target`



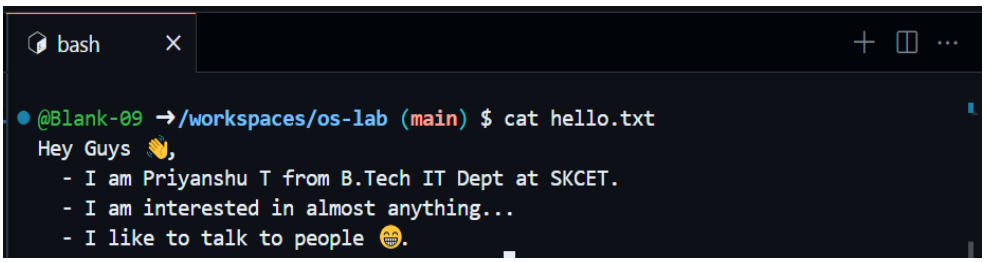
7. **Command:** cp
   Description: Opens the vim text editor.
   Syntax: `cp source destination`



8. **Command:** vi
   Description: Opens the vim text editor.
   Syntax: `vi [filename]`



9. **Command:** cat
   Description: Displays the contents of files.
   Syntax: `cat [file]`



10. **Command:** rm
    Description: Deletes files or directories.
    Syntax: `rm [options] file`

Name:
Reg. No:

```
bash lab_01
@Blank-09 →/workspaces/os-lab/lab_01 (main) $ ls
hello.txt
@Blank-09 →/workspaces/os-lab/lab_01 (main) $ rm hello.txt
@Blank-09 →/workspaces/os-lab/lab_01 (main) $ ls
@Blank-09 →/workspaces/os-lab/lab_01 (main) $
```

11. **Command:** clear
    Description: Clears the terminal screen.
    Syntax: `clear`

```
bash lab_01
@Blank-09 →/workspaces $ cd lab_01
@Blank-09 →/workspaces/lab_01 $ clear
```

```
@Blank-09 →/workspaces/lab_01 $
```

◀ Result

12. **Command:** sudo
    Description: Executes a command with superuser (root) privileges.
    Syntax: `sudo [options] command`

```
bash lab_01
@Blank-09 →/workspaces/lab_01 $ sudo apt-get update
Get:1 https://packages.microsoft.com/repos/microsoft-ubuntu-focal-prod focal InRelease [3632 B]
Get:2 https://dl.yarnpkg.com/debian stable InRelease [17.1 kB]
Get:3 https://repo.anaconda.com/pkgs/misc/debrepo/conda stable InRelease [3961 B]
Get:4 https://packages.microsoft.com/repos/microsoft-ubuntu-focal-prod focal/main all Packages [2714 B]
Get:5 https://packages.microsoft.com/repos/microsoft-ubuntu-focal-prod focal/main amd64 Packages [288 kB]
Get:6 http://security.ubuntu.com/ubuntu focal-security InRelease [114 kB]
Get:7 https://dl.yarnpkg.com/debian stable/main all Packages [11.8 kB]
Get:8 https://dl.yarnpkg.com/debian stable/main amd64 Packages [11.8 kB]
Get:9 https://repo.anaconda.com/pkgs/misc/debrepo/conda stable/main amd64 Packages [4557 B]
Get:10 http://archive.ubuntu.com/ubuntu focal InRelease [265 kB]
Get:12 http://security.ubuntu.com/ubuntu focal-security/multiverse amd64 Packages [29.8 kB]
Get:13 http://security.ubuntu.com/ubuntu focal-security/main amd64 Packages [3672 kB]
Get:14 http://archive.ubuntu.com/ubuntu focal-updates InRelease [114 kB]
Get:15 http://archive.ubuntu.com/ubuntu focal-backports InRelease [108 kB]
Get:16 http://archive.ubuntu.com/ubuntu focal/multiverse amd64 Packages [177 kB]
Get:11 https://packagecloud.io/github/git-lfs/ubuntu focal InRelease [28.0 kB]
Get:17 http://archive.ubuntu.com/ubuntu focal/restricted amd64 Packages [33.4 kB]
Get:18 http://archive.ubuntu.com/ubuntu focal/universe amd64 Packages [11.3 MB]
Get:19 http://security.ubuntu.com/ubuntu focal-security/universe amd64 Packages [1207 kB]
Get:20 http://security.ubuntu.com/ubuntu focal-security/restricted amd64 Packages [3616 kB]
Get:22 http://archive.ubuntu.com/ubuntu focal/main amd64 Packages [1275 kB]
Get:23 http://archive.ubuntu.com/ubuntu focal-updates/main amd64 Packages [4147 kB]
Get:24 http://archive.ubuntu.com/ubuntu focal-updates/universe amd64 Packages [1503 kB]
Get:25 http://archive.ubuntu.com/ubuntu focal-updates/restricted amd64 Packages [3766 kB]
Get:21 https://packagecloud.io/github/git-lfs/ubuntu focal/main amd64 Packages [3690 B]
Get:26 http://archive.ubuntu.com/ubuntu focal-updates/multiverse amd64 Packages [32.5 kB]
Get:27 http://archive.ubuntu.com/ubuntu focal-backports/universe amd64 Packages [28.6 kB]
Get:28 http://archive.ubuntu.com/ubuntu focal-backports/main amd64 Packages [55.2 kB]
Fetched 31.9 MB in 5s (6690 kB/s)
Reading package lists... Done
```

13. **Command:** chmod
    Description: Changes the file mode (permissions) of a file or directory.
    Syntax: `chmod [options] mode file`

```
bash lab_01
@Blank-09 →/workspaces/lab_01 $ ls -l
total 0
-rw-rw-rw- 1 codespace codespace 0 May 25 10:17 script.sh
@Blank-09 →/workspaces/lab_01 $ chmod 755 script.sh
@Blank-09 →/workspaces/lab_01 $ ls -l
total 0
-rwxr-xr-x 1 codespace codespace 0 May 25 10:17 script.sh
```

14. **Command:** zip
    Description: Compresses files into a ZIP archive.
    Syntax: `zip [options] zipfile file1 file2 ...`

Name:
Reg. No:

```
bash lab_01
@Blank-09 →/workspaces/lab_01 $ ls
file1.txt  file2.txt
@Blank-09 →/workspaces/lab_01 $ zip archive.zip file1.txt file2.txt
  adding: file1.txt (deflated 58%)
  adding: file2.txt (deflated 85%)
@Blank-09 →/workspaces/lab_01 $ ls
archive.zip  file1.txt  file2.txt
```

15. **Command:** unzip
    Description: Extracts files from a ZIP archive.
    Syntax: `unzip [options] zipfile`

```
bash lab_01
@Blank-09 →/workspaces/lab_01 $ ls
archive.zip
@Blank-09 →/workspaces/lab_01 $ unzip archive.zip
Archive:  archive.zip
  inflating: file1.txt
  inflating: file2.txt
@Blank-09 →/workspaces/lab_01 $ ls
archive.zip  file1.txt  file2.txt
```

16. **Command:** echo
    Description: Displays a line of text.
    Syntax: `echo [options] [STRING]`

```
@G-Pavithran-dev →/workspaces/codespaces-blank $ echo "hello world"
hello world
@G-Pavithran-dev →/workspaces/codespaces-blank $
```

17. **Command:** man
    Description: An interface to the system reference manuals.
    Syntax: `man [man option]`

```
@G-Pavithran-dev →/workspaces/codespaces-blank $ man echo
ECHO(1)                          User Commands

NAME
       echo - display a line of text

SYNOPSIS
       echo [SHORT-OPTION]... [STRING]...
       echo LONG-OPTION

DESCRIPTION
       Echo the STRING(s) to standard output.

       -n     do not output the trailing newline
```

18. **Command:** whoami
    Description: Print effective userID.
    Syntax: `whoami [OPTION]`

```
@G-Pavithran-dev →/workspaces/codespaces-blank $ whoami
codespace
@G-Pavithran-dev →/workspaces/codespaces-blank $
```

19. **Command:** ps
    Description: Report a snapshot of the current processes.
    Syntax: `ps [OPTIONS]`

```
@G-Pavithran-dev →/workspaces/codespaces-blank $ ps
    PID TTY          TIME CMD
   1401 pts/0    00:00:00 bash
   5320 pts/0    00:00:00 ps
@G-Pavithran-dev →/workspaces/codespaces-blank $
```

20. **Command:** sort
    Description: Sort lines of text files.
    Syntax: `sort [OPTION] [FILE]`

```
@G-Pavithran-dev →/workspaces/codespaces-blank/linux $ cat sample.txt
Hi, I'm Pavithran
Linux
Bash
Windows
@G-Pavithran-dev →/workspaces/codespaces-blank/linux $ sort sample.txt
Bash
Hi, I'm Pavithran
Linux
Windows
```

21. **Command:** df
    Description: Report file system disk space usage.
    Syntax: `df [OPTION] [FILE]`

```
@G-Pavithran-dev →/workspaces/codespaces-blank/linux $ df
Filesystem     1K-blocks     Used Available Use% Mounted on
overlay         32847680 16055876  15097704  52% /
tmpfs              65536        0     65536   0% /dev
shm                65536        0     65536   0% /dev/shm
/dev/root       30298176 24470680   5811112  81% /vscode
/dev/sdb1       46127956      128  43752252   1% /tmp
/dev/loop3      32847680 16055876  15097704  52% /workspaces
```

22. **Command:** ip
    Description: Show / manipulate routing, network devices, interfaces and tunnels.
    Syntax: `ip [options]`

```
@G-Pavithran-dev →/workspaces/codespaces-blank/linux $ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
```

23. **Command:** wc
    Description: Print newline, word, and byte counts for each file.
    Syntax: `wc [OPTION] [FILE]`

```
@G-Pavithran-dev →/workspaces/codespaces-blank/linux $ wc sample.txt
 4  6 37 sample.txt
```

24. **Command:** cal
    Description: Displays a calendar and the date of Easter.
    Syntax: `cal [OPTIONS]`

Name:
Reg. No:

```
@G-Pavithran-dev →/workspaces/codespaces-blank/linux $ cal
      May 2024
Su Mo Tu We Th Fr Sa
          1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31
```

25. **Command:** head
    Description: Output the first part of files.
    Syntax: `head [options] [file]`

```
@G-Pavithran-dev →/workspaces/codespaces-blank/linux $ head sample.txt
Hi, I'm Pavithran
Linux
macOS
FreeBSD
OpenBSD
NetBSD
Windows XP
Windows Vista
Windows 7
Windows 8
```

26. **Command:** tail
    Description: Output the last part of files.
    Syntax: `tail [options] [file]`

```
@G-Pavithran-dev →/workspaces/codespaces-blank/linux $ tail sample.txt
Windows 8.1
Windows 10
Windows 11
Chromium OS
Android
Apple macOS
Google's Android OS
Microsoft Windows
Apple iOS
Linux Operating System
```

27. **Command:** diff
    Description: Compare files line by line.
    Syntax: `diff [options] files...`

```
@G-Pavithran-dev →/workspaces/codespaces-blank/linux $ diff sample.txt samp
4,7c4,7
< mac chip 2
< Windows 11
< Chromium OS 198
< Android
---
> Ubuntu
> Windows 10
> Chromium OS
> Android 14
```

28. **Command:** cmp

Description: Compare two files byte by byte.
Syntax: `cmp [OPTION] FILE...`

```
@G-Pavithran-dev ➜/workspaces/codespaces-blank/linux $ cmp sample.txt sample1.txt
sample.txt sample1.txt differ: byte 40, line 4
```

29. **Command:** comm
Description: Compare two sorted files line by line.
Syntax: `comm [OPTION] FILE1 FILE2`

```
@G-Pavithran-dev ➜/workspaces/codespaces-blank/linux $ comm sample.txt sample1.txt
                Hi, I'm Pavithran
                Linux
                macOS
        Windows 10
comm: file 2 is not in sorted order
        Chromium OS
        Android 14
Windows 11
comm: file 1 is not in sorted order
Chromium OS
Android
```

30. **Command:** apt (for ubuntu)
Description: Command-Line interface provided for Ubuntu
Syntax: `apt [options] {get | list | update}`

```
root ➜/workspaces/codespaces-blank/linux $ apt update
Get:1 https://packages.microsoft.com/repos/microsoft-ubuntu-focal-prod focal InRelease [3632 B]
Get:2 https://dl.yarnpkg.com/debian stable InRelease [17.1 kB]
Get:3 https://repo.anaconda.com/pkgs/misc/debrepo/conda stable InRelease [3961 B]
Get:4 https://packages.microsoft.com/repos/microsoft-ubuntu-focal-prod focal/main all Packages [2714 B]
Get:5 https://packages.microsoft.com/repos/microsoft-ubuntu-focal-prod focal/main amd64 Packages [288 kB]
Get:6 https://dl.yarnpkg.com/debian stable/main all Packages [11.8 kB]
Get:7 https://dl.yarnpkg.com/debian stable/main amd64 Packages [11.8 kB]
Get:8 https://repo.anaconda.com/pkgs/misc/debrepo/conda stable/main amd64 Packages [4557 B]
Get:9 http://security.ubuntu.com/ubuntu focal-security InRelease [114 kB]
Get:10 http://archive.ubuntu.com/ubuntu focal InRelease [265 kB]
Get:12 http://security.ubuntu.com/ubuntu focal-security/restricted amd64 Packages [3616 kB]
Get:11 https://packagecloud.io/github/git-lfs/ubuntu focal InRelease [28.0 kB]
Get:13 http://archive.ubuntu.com/ubuntu focal-updates InRelease [114 kB]
Get:14 http://archive.ubuntu.com/ubuntu focal-backports InRelease [108 kB]
Get:15 http://archive.ubuntu.com/ubuntu focal/universe amd64 Packages [11.3 MB]
Get:17 http://security.ubuntu.com/ubuntu focal-security/main amd64 Packages [3672 kB]
Get:18 http://security.ubuntu.com/ubuntu focal-security/universe amd64 Packages [1207 kB]
Get:19 http://security.ubuntu.com/ubuntu focal-security/multiverse amd64 Packages [29.8 kB]
Get:16 https://packagecloud.io/github/git-lfs/ubuntu focal/main amd64 Packages [3690 B]
Get:20 http://archive.ubuntu.com/ubuntu focal/restricted amd64 Packages [33.4 kB]
Get:21 http://archive.ubuntu.com/ubuntu focal/main amd64 Packages [1275 kB]
Get:22 http://archive.ubuntu.com/ubuntu focal/multiverse amd64 Packages [177 kB]
Get:23 http://archive.ubuntu.com/ubuntu focal-updates/restricted amd64 Packages [3766 kB]
Get:24 http://archive.ubuntu.com/ubuntu focal-updates/multiverse amd64 Packages [32.5 kB]
Get:25 http://archive.ubuntu.com/ubuntu focal-updates/universe amd64 Packages [1503 kB]
```

RESULT:

The basic Linux commands are successfully executed and output is verified.

Name:
Reg. No:

Ex no: 02

Date: 06.05.2024

PROGRAMS USING SHELL PROGRAMMING

AIM:

To write a programs using shell programming.

DESCRIPTION:

A Linux shell is a command language interpreter, the primary purpose of which is to translate the command lines typed at the terminal into system actions. The shell itself is a program, through which other programs are invoked

What is a shell script?

- A shell script is a file containing a list of commands to be executed by the Linux shell. shell script provides the ability to create your own customized Linux commands
- Linux shell have sophisticated programming capabilities which makes shell script powerful Linux tools

SYNTAX

1. EXPRESSION Command:

To perform all arithmetic operations.

Syntax: var = „expr $value1 + $value2" or
var = $(($value1 + $value2)) Example:
a=10 b=20 sum=$(($a + $b)) echo $sum

2. OPERATORS:

Shell uses the built-in test command operators to test numbers and strings.

Equality:

= string

!= string

-eq number

-ne number

ARITHMETIC OPERATORS OPERATOR	**DESCRIPTION**	**EXAMPLE**
+ Addition	Adds value on either side of the operator	`expr $a + $b` will give 30
-Subtraction	Subtracts right hand operand from left hand operand	`expr $a - $b` will give -10
* (Multiplication)	Multiplies values on either side of the operator	`expr $a \* $b` will give 200
/ (Division)	Divides left hand operand by right hand operand	`expr $b / $a` will give 2
% (Modulus)	Divides left hand operand by right hand operand and returns remainder	`expr $b % $a` will give 0

Name:
Reg. No:

= (Assignment)	Assigns right operand in left operand	a = $b would assign value of b into a

RELATIONAL OPERATORS OPERATOR	**DESCRIPTION**	**EXAMPLE**
-eq	Checks if the value of two operands are equal or not; if yes, then the condition becomes true.	[$a -eq $b] is not true.
-ne	Checks if the value of two operands are equal or not; if values are not equal, then the condition becomes true.	[$a -ne $b] is true.
-gt	Checks if the value of left operand is greater than the value of right operand; if yes, then the condition becomes true.	[$a -gt $b] is not true.
-lt	Checks if the value of left operand is less than the value of right operand; if yes, then the condition becomes true.	[$a -lt $b] is true.
-ge	Checks if the value of left operand is greater than or equal to the value of right	[$a -ge $b] is not true

	operand; if yes, then the condition becomes true.	
-le	Checks if the value of left operand is less than or equal to the value of right operand; if yes, then the condition becomes true.	[$a -le $b] is true.
BOOLEAN OPERATORS OPERATOR	**DESCRIPTION**	**EXAMPLE**
!	This is logical negation. This inverts a true condition into false and vice versa.	[! false] is true.
-o	This is logical **OR**. If one of the operands is true, then the condition becomes true.	[$a -lt 20 -o $b -gt 100] is true

-a	This is logical **AND**. If both the operands are true, then the condition becomes true otherwise false.	[$a -lt 20 -a $b -gt 100] is false.

STRING OPERATORS **OPERATOR**	**DESCRIPTION**	**EXAMPLE**
=	Checks if the value of two operands are equal or not; if yes, then the condition becomes true.	[$a = $b] is not true
!=	Checks if the value of two operands are equal or not; if values are not equal then the condition becomes true.	[$a != $b] is true.
-z	Checks if the given string operand size is zero; if it is zero length, then it returns true.	[-z $a] is not true.
-n	Checks if the given string operand size is non-zero; if it is nonzero length, then it returns true.	[-n $a] is not false.
str	Checks if **str** is not the empty string; if it is empty, then it returns false.	[$a] is not false.

3. DESCISION MAKING STATEMENTS

The **if...else** statements

Example if [$a -gt $b] then
echo “$a is Big”

else

echo “$b is Big”

fi

The **case...esac** statement **Example**

echo "1.print 2.Exit" read op
case $op in 1)echo “Hello”;;
2)exit esac

4. LOOPING STATEMENTS

for loop statements

Example 1 for i in 1 2 do
echo “welcome” done

Example 2 n=10 for ((i=0; i<n;
i++)) do echo $i done

Name:
Reg. No:

while loop statements

Example a=0 while [$a -lt 10
] do echo $a a=`expr $a + 1`
done

Note:

i) echo –n

-n option lets echo avoid printing a new line character. **ii)** To calculate more than two numbers use echo Example: x=3 y=6 z=9
echo "$x+$y+$z" | bc

PROGRAMS:

1. **Write a Shell program to check the given number is even or odd.**

   Sample Input :

   Enter any number

   23

   Sample Output : 23 is odd

Result:

```
echo "Enter a number: "
read n
rem=$(($n % 2))
if [$rem -eq 0]
then
        echo "$n is even number"
else
        echo "$n is odd number"
fi
~
~
~
~
~
~
"oddoreven.sh" [New] 9L, 127B written
[root@localhost ~]# sh oddoreven.sh
Enter a number:
23
23 is odd number
[root@localhost ~]#
```

2. **Write a Shell program to check the given number and its reverse are same.**

   Sample Input :

   Enter a number

   252

   Sample Output : The given number and its reverse are same

Result:

Name:
Reg. No:

```
echo "Enter the number: "
read n
number=$n
reverse=0
while [$n -gt 0]
do
a = `expr $n %10`
n = `expr $n / 10`
reverse = `expr $reverse \*10 + $a `
done
echo $reverse
if [$number -eq $reverse]
then
echo "The Number is same as the Reverse"
else
echo "The Number is NOT same as the Reverse"
fi
~
~
~
~
~
~
[root@localhost ~]# sh reverse.sh
Enter the number:
12345
The Number is NOT same as the Reverse
[root@localhost ~]#
```

3. **Write a Shell Program to find a factorial of a number.**

Sample Input :

Enter number

5

Sample Output : 120

Result:

```
read -p "Enter a number: "
fact=1
while [$num -gt 1]
do
fact=$((fact*num))
num=$((num-1))
done
echo $fact

~
~
~
~
~
~
~
~
"fact.sh" [New] 9L, 107B written
[root@localhost ~]# sh fact.sh
Enter a number: 4
 24
```

4. **Write a Shell Program for finding sum of Odd and Even numbers up to 'N'.**

Sample Input :

Enter the number of elements

5

Name:
Reg. No:

Enter the number

23

34

45

56

67

Sample Output :

The sum of odd numbers is : 135

The sum of even numbers is : 90

Result:

```
read -p "Enter a number: "
fact=1
while [$num -gt 1]
do
fact=$((fact*num))
num=$((num-1))
echo "Enter n value: "
read n
sumodd=0
sumeven=0
i=0
while [&i -ne $n]
do
echo "Enter Number: "
read num
if [`expr $num % 2` -ne 0]
then
sumodd = `expr $sumodd + $num`
sumeven = `expr $sumeven + $num`
fi
i = `expr $i +1`
done
echo "Sum of odd numbers = $sumodd"
echo "Sum of even numbers = $sumeven"
```

```
[root@localhost ~]# sh fifty.sh
Enter n value:
5
Enter Number:
23
34
45
56
67
Sum of odd numbers = 135
Sum of even numbers = 90
```

5. Write a Shell program to display student grades. Sample Output :

Roll no.	Name	Total	Average	Grade
19skcet001	Anil	201	67	First class

Result:

```
echo "Enter Student name :"
read name
echo "Enter the Reg.no :"
read rno
echo "Enter Mark1 :"
read m1
echo "Enter Mark2 :"
read m2
echo "Enter Mark3 :"
read m3
tot = $(expr $m1 + $m2 + $m3)
avg = $(expr $tot/3)
echo "Student Name : $name"
echo "Reg.no : $rno"
echo "Total : $tot"
echo "Average : $avg"
if [$m1 -ge 65] && [$m2 -ge 65] && [m3 -ge 65]
then
echo "Grade : First Class"
else
echo "Grade : NOT First Class"
~
~
~
~
~
~
"student.sh" [New] 21L, 418B written
```

```
[root@localhost ~]# sh student.sh
Enter Student name :
Arthika
Enter the Reg.no :
20EUCS015
Enter Mark1 :
75
Enter Mark2 :
85
Enter Mark3 :
95
Student Name : Arthika
Reg.no : 20EUCS015
Total : 255
Average : 85
Grade : First Class
```

6. Write a Shell Program to have to print half pyramid using for loop.

Sample Input : n=10

Sample Output:

```
*

* *

* * *

* * * *

* * * * *

* * * * * *

* * * * * * *

* * * * * * * *

* * * * * * * * *

* * * * * * * * * *
```

Result:

```
echo "Enter a number :"
read rows
for ((i=1 ; i<=rows ; i++))
do
 for ((j=1 ; j<=i ; j++))
  do
   echo -n "* "
  done
 echo
done
~
~
~
~
~
~
~
~
"pattern.sh" [New] 10L, 130B written
[root@localhost ~]# sh pattern.sh
Enter a number :
10
*
* *
* * *
* * * *
* * * * *
* * * * * *
* * * * * * *
* * * * * * * *
* * * * * * * * *
* * * * * * * * * *
[root@localhost ~]#
```

Name:
Reg.no:

7. Write a Shell Program to perform Arithmetic Operation using Case statement.

Sample Input :

Enter two number

2 3

1. Add 2.Sub 3.Mul 4.Div 5.Exit

Enter the option: 1 **Sample Output :**

Addition is 5

Result:

```
echo "Enter 2  numbers :"
read a b
echo "Enter Your Choice :"
echo "1)Add 2)Sub 3)Mul 4)Div 5)Exit"
read n
case "$n" in
        1) echo "Sum : `expr $a + $b`";;
        2) echo "Difference : `expr $a - $b`";;
        3) echo "Product : `expr $a \* $b`";;
        4) echo "Quotient : `expr $a / $b`";;
esac
~
~
~
~
~
~
~
~
~
~
~
~
~
"menu.sh" 11L, 278B written
[root@localhost ~]# sh menu.sh
Enter 2  numbers :
2 3
Enter Your Choice :
1)Add 2)Sub 3)Mul 4)Div 5)Exit
3
Product : 6
[root@localhost ~]#
```

8. Write a Shell Program for comparison of strings.

Sample Input :

Enter string 1: OS

Enter string 2: OS **Sample Output :**

String 1 and String 2 are identical **Result:**

Name:
Reg.no:

```
read -p "Enter 1st Sring :" VAR1
read -p "Enter 2nd String :" VAR2
if [["$VAR1" == "$VAR2"]];
then
echo "Strings are Equal"
else
echo "Strings are Not Equal"
fi
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"equal.sh" [New] 8L, 161B written
[root@localhost ~]# sh equal.sh
Enter 1st Sring :STRING
Enter 2nd String :STRING
Strings are Equal
[root@localhost ~]#
```

9. **Write a Shell program to find the smallest number from a set of numbers.**

   Sample Input :

   Enter the number of elements: 5 Enter the numbers:

   34

   23

   45

   37

   56

   Sample Output :

```
echo "Enter the No.of Elements :"
read n
for ((i=0 ; i<n ; i++))
        do
                read nos[$i]
        done
if [${nos[$i]} -lt $small ]; then
        small = ${nos[$i]}
fi

echo "Smallest Number : $small"
~
```

```
[root@localhost ~]# sh small.sh
Enter the No.of Elements :
3
21
34
5
Smallest Number : 5
[root@localhost ~]#
```

10. **Write a Shell program to find the sum of all numbers between 50 and 100, which are divisible by 3 and not divisible by 5.**

Name:
Reg.no:

Sample Output :

The sum of all numbers between 50 and 100, which are divisible by 3 and not divisible by 5 are:

51

54

57

63

66

69

72

78

81

84

87

93

96

99

Output:

```
for ((i=50 ; i<=100 ; i++)) do
        if [`expr $i %3` = 0 -a `expr $i % 5` !=0 ] then
                echo $i
        fi
done
```

```
[root@localhost ~]# sh sum1.sh
51
54
57
63
66
69
72
78
81
84
87
93
96
99
[root@localhost ~]#
```

RESULT

Thus the programs using Shell programming is successfully executed.

Name:
Reg.no:

Ex no: 03
Date: 07.05.2024

Use of case statements and Menu driven program

AIM:

To illustrate the use of switch statement in Menu driven using shell programming.

ALGORITHM:

1. Start
2. Read the choice from the user
3. Use switch case statement to do the operation
   3.1 If choice is 1 perform Fibonacci series
   3.2 If choice is 2 write a shell program to check whether a given number is odd or even
   3.3 If choice is 3, write a shell program to check whether a given year is Leap year or not
   3.4 If choice is 4 write a shell program to find the greatest of three numbers
   3.5 If choice is 5, write a shell program to find the sum of the digits of a given number

2. Menu Driven program for file operations:

1. Start
2. Using while loop, perform
3. Get the choice from user
4. Create switch case to perform
   4.1 If choice is 1, perform cat command
   4.2 If choice is 1, perform cp command
   4.3 If choice is 3, perform mv command
   4.4 If choice is 4, perform wc command
   4.5 If choice is 5, perform grep command
   4.6 If choice is 6, perform head command
   4.7 If choice is 7, perform tail command
   4.8 If choice is 8, perform sort command

PROGRAM:

(i) Driver Menu

```
#!/bin/bash

echo "Choose an option:"
echo "1. Perform Fibonacci series"
echo "2. Check whether a given number is odd or even"
echo "3. Check whether a given year is a Leap year or not"
echo "4. Find the greatest of three numbers"
echo "5. Find the sum of the digits of a given number"
echo
echo "Enter choice: "
read choice
echo

case $choice in
  1)
    echo "Performing Fibonacci series..."
```

Name:
Reg.no:

```
  echo "Enter the number of terms for Fibonacci series:"
  read n
  a=0
  b=1
  echo "The Fibonacci series is:"
  for (( i=0; i<n; i++ ))
  do
    echo -n "$a "
    fn=$((a + b))
    a=$b
    b=$fn
  done
  echo
  ;;
2)
  echo "Checking whether a given number is odd or even..."
  echo "Enter a number:"
  read number
  if (( number % 2 == 0 ))
  then
    echo "$number is even."
  else
    echo "$number is odd."
  fi
  ;;
3)
  echo "Checking whether a given year is a Leap year or not..."
  echo "Enter a year:"
  read year
  if (( (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0) ))
  then
    echo "$year is a leap year."
  else
    echo "$year is not a leap year."
  fi
  ;;
4)
  echo "Finding the greatest of three numbers..."
  echo "Enter three numbers:"
  read num1
  read num2
  read num3
  if (( num1 >= num2 && num1 >= num3 ))
  then
    echo "$num1 is the greatest."
  elif (( num2 >= num1 && num2 >= num3 ))
  then
    echo "$num2 is the greatest."
  else
    echo "$num3 is the greatest."
  fi
  ;;
5)
```

```
    echo "Finding the sum of the digits of a given number..."
    echo "Enter a number:"
    read number
    sum=0
    while [ $number -gt 0 ]
    do
      digit=$((number % 10))
      sum=$((sum + digit))
      number=$((number / 10))
    done
    echo "The sum of the digits is $sum."
    ;;
  *)
    echo "Invalid choice. Please enter a number between 1 and 5."
    ;;
esac
```

(ii) Command Menu

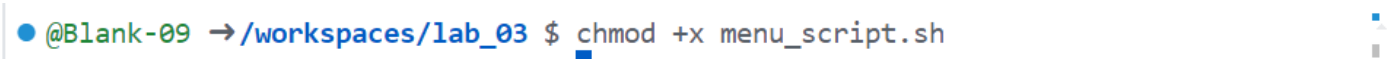
```
#!/bin/bash

while true; do
    echo "Choose an option:"
    echo "1. Perform cat command"
    echo "2. Perform cp command"
    echo "3. Perform mv command"
    echo "4. Perform wc command"
    echo "5. Perform grep command"
    echo "6. Perform head command"
    echo "7. Perform tail command"
    echo "8. Perform sort command"
    echo "9. Exit"

    read choice

    case $choice in
        1)
            echo "Enter the filename for cat command:"
            read filename
            cat "$filename"
            ;;
        2)
            echo "Enter the source file for cp command:"
            read srcfile
            echo "Enter the destination file for cp command:"
            read destfile
            cp "$srcfile" "$destfile"
            ;;
        3)
            echo "Enter the source file for mv command:"
            read srcfile
            echo "Enter the destination file for mv command:"
            read destfile
```

Name:
Reg.no:

```
            mv "$srcfile" "$destfile"
            ;;
        4)
            echo "Enter the filename for wc command:"
            read filename
            wc "$filename"
            ;;
        5)
            echo "Enter the pattern to search for with grep command:"
            read pattern
            echo "Enter the filename for grep command:"
            read filename
            grep "$pattern" "$filename"
            ;;
        6)
            echo "Enter the filename for head command:"
            read filename
            head "$filename"
            ;;
        7)
            echo "Enter the filename for tail command:"
            read filename
            tail "$filename"
            ;;
        8)
            echo "Enter the filename for sort command:"
            read filename
            sort "$filename"
            ;;
        9)
            echo "Exiting..."
            break
            ;;
        *)
            echo "Invalid choice. Please enter a number between 1 and 9."
            ;;
    esac
done
```

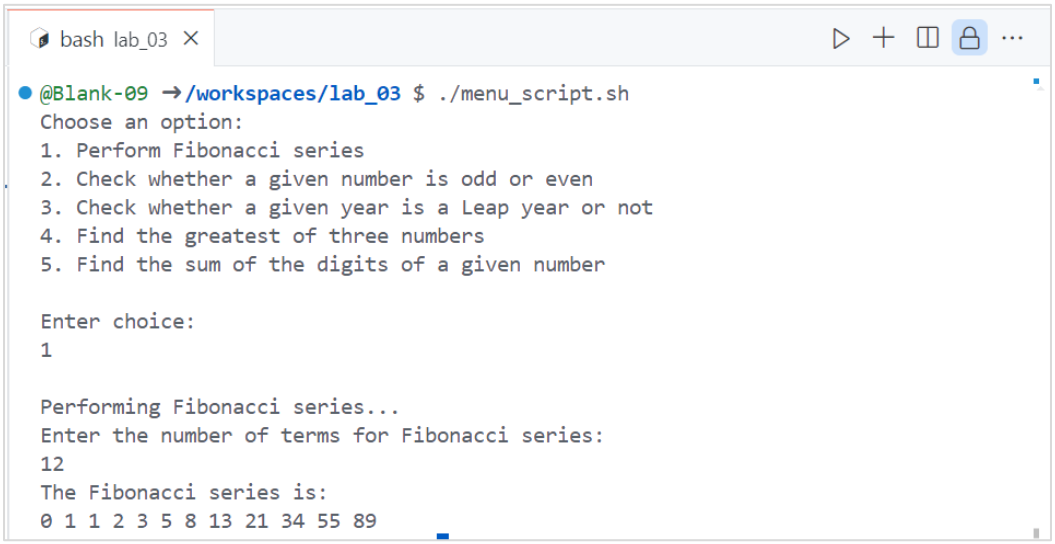
OUTPUT:

Run this before executing any bash script

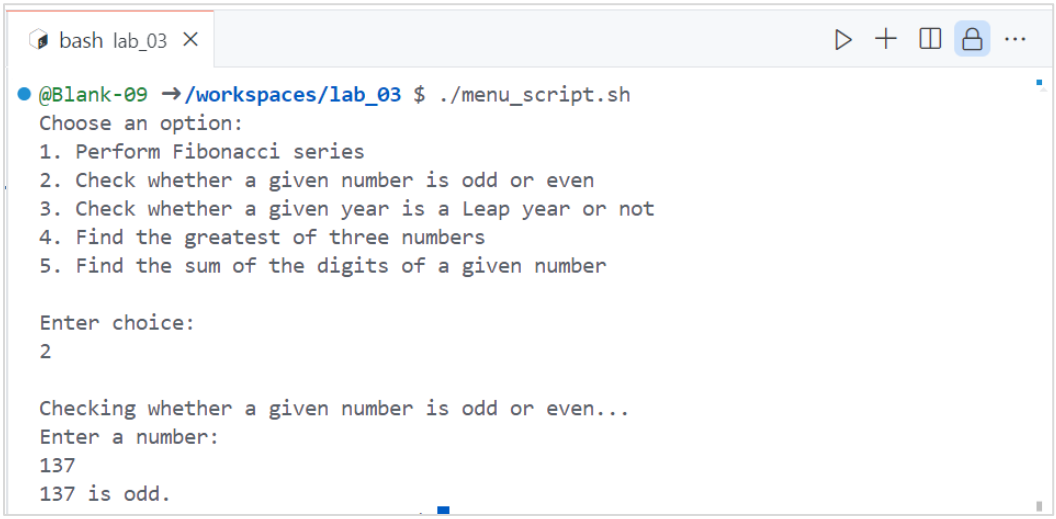


Name:
Reg.no:

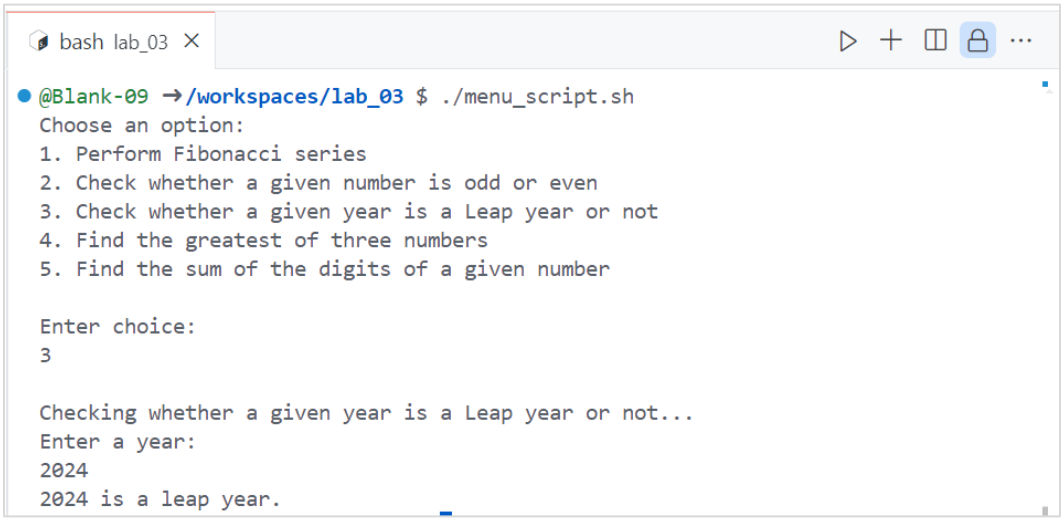
Fibonnaci Series:



Odd or Even:



Leap Year:



Greatest of three number:

Name:
Reg.no:

```
bash lab_03
@Blank-09 →/workspaces/lab_03 $ ./menu_script.sh
Choose an option:
1. Perform Fibonacci series
2. Check whether a given number is odd or even
3. Check whether a given year is a Leap year or not
4. Find the greatest of three numbers
5. Find the sum of the digits of a given number

Enter choice:
4

Finding the greatest of three numbers...
Enter three numbers:
92
137
108
137 is the greatest.
```

Sum of digits:

```
bash lab_03
@Blank-09 →/workspaces/lab_03 $ ./menu_script.sh
Choose an option:
1. Perform Fibonacci series
2. Check whether a given number is odd or even
3. Check whether a given year is a Leap year or not
4. Find the greatest of three numbers
5. Find the sum of the digits of a given number

Enter choice:
5

Finding the sum of the digits of a given number...
Enter a number:
137
The sum of the digits is 11.
```

CAT:

```
bash lab_03
@Blank-09 →/workspaces/lab_03 $ ./command_menu.sh
Choose an option:
1. Perform cat command
2. Perform cp command
3. Perform mv command
4. Perform wc command
5. Perform grep command
6. Perform head command
7. Perform tail command
8. Perform sort command
9. Exit

Enter choice:
1

Enter the filename for cat command:
menu_script.sh
#!/bin/bash

echo "Choose an option:"
echo "1. Perform Fibonacci series"
echo "2. Check whether a given number is odd or even"
```

CP:

Name:
Reg.no:

```
bash lab_03
@Blank-09 →/workspaces/lab_03 $ ./command_menu.sh
Choose an option:
1. Perform cat command
2. Perform cp command
3. Perform mv command
4. Perform wc command
5. Perform grep command
6. Perform head command
7. Perform tail command
8. Perform sort command
9. Exit

Enter choice:
2

Enter the source file for cp command:
menu_script.sh
Enter the destination file for cp command:
cloned_script.sh
```

MV:

```
bash lab_03
@Blank-09 →/workspaces/lab_03 $ ./command_menu.sh
Choose an option:
1. Perform cat command
2. Perform cp command
3. Perform mv command
4. Perform wc command
5. Perform grep command
6. Perform head command
7. Perform tail command
8. Perform sort command
9. Exit

Enter choice:
3

Enter the source file for mv command:
new_script.sh
Enter the destination of file for mv command:
moved_folder
```

WC:

```
bash lab_03
@Blank-09 →/workspaces/lab_03 $ ./command_menu.sh
Choose an option:
1. Perform cat command
2. Perform cp command
3. Perform mv command
4. Perform wc command
5. Perform grep command
6. Perform head command
7. Perform tail command
8. Perform sort command
9. Exit

Enter choice:
4

Enter the filename for wc command:
menu_script.sh
  85  317 1901 menu_script.sh
```

GREP:

Name:
Reg.no:

bash lab_03

```
1. Perform cat command
2. Perform cp command
3. Perform mv command
4. Perform wc command
5. Perform grep command
6. Perform head command
7. Perform tail command
8. Perform sort command
9. Exit

Enter choice:
5

Enter the filename for grep command:
command_menu.sh
Enter the pattern to search for with grep command:
Choose
    echo "Choose an option:"
```

HEAD:

bash lab_03

```
@Blank-09 →/workspaces/lab_03 $ ./command_menu.sh
Choose an option:
1. Perform cat command
2. Perform cp command
3. Perform mv command
4. Perform wc command
5. Perform grep command
6. Perform head command
7. Perform tail command
8. Perform sort command
9. Exit

Enter choice:
6

Enter the filename for head command:
menu_script.sh
#!/bin/bash

echo "Choose an option:"
echo "1. Perform Fibonacci series"
echo "2. Check whether a given number is odd or even"
```

TAIL:

bash lab_03

```
@Blank-09 →/workspaces/lab_03 $ ./command_menu.sh
Choose an option:
1. Perform cat command
2. Perform cp command
3. Perform mv command
4. Perform wc command
5. Perform grep command
6. Perform head command
7. Perform tail command
8. Perform sort command
9. Exit

Enter choice:
7

Enter the filename for tail command:
menu_script.sh
      digit=$((number % 10))
      sum=$((sum + digit))
      number=$((number / 10))
    done
```

SORT:

Name:
Reg.no:

bash lab_03

```
@Blank-09 →/workspaces/lab_03 $ ./command_menu.sh
Choose an option:
1. Perform cat command
2. Perform cp command
3. Perform mv command
4. Perform wc command
5. Perform grep command
6. Perform head command
7. Perform tail command
8. Perform sort command
9. Exit

Enter choice:
8

Enter the filename for sort command:
menu_script.sh


      a=$b
      b=$fn
      digit=$((number % 10))
      echo "$num1 is the greatest."
      echo "$num2 is the greatest."
      echo "$num3 is the greatest."
      echo "$number is even."
      echo "$number is odd."
      echo "$year is a leap year."
      echo "$year is not a leap year."
      echo -n "$a "
      fn=$((a + b))
```

RESULT:

The shell program is successfully executed. And the output is verified by the sample output.

Name:
Reg.no:

Ex no: 04
Date: 08.05.2024

Implementation of FCFS and SJF CPU scheduling algorithms

AIM:

To write a C++ program to implement the FCFS and SJF CPU Scheduling algorithms.

ALGORITHM:

1. Start the program.
2. Declare the variable.
3. Input the number of processes from user.
4. Create 'for' loop to input arrival time and burst time.
5. Using 'for' loop, calculate:
   i. Turn Around Time = Completion Time – Arrival Time
   ii. Waiting Time =Turn Around Time – Burst Time
6. Calculate Average Turn Around Time and Average Waiting time.
7. Print the results.
8. Stop the program.

PROGRAM:

CODE for FCFS:

```
#include<iostream>
using namespace std;

void findWaitingTime(int processes[], int n,int bt[], int wt[])
{
   wt[0] = 0;
   for (int i = 1; i < n ; i++ )
      wt[i] = bt[i-1] + wt[i-1] ;
}

void findTurnAroundTime( int processes[], int n,int bt[], int wt[], int tat[])
{
   for (int i = 0; i < n ; i++)
      tat[i] = bt[i] + wt[i];
}

void findavgTime( int processes[], int n, int bt[])
{
   int wt[n], tat[n], total_wt = 0, total_tat = 0;
   findWaitingTime(processes, n, bt, wt);
   findTurnAroundTime(processes, n, bt, wt, tat);
   cout << "Processes "<< " Burst time " << " Waiting time " << " Turn around time\n";
   for (int i=0; i<n; i++)
   {
      total_wt = total_wt + wt[i];
      total_tat = total_tat + tat[i];
      cout << "    " << i+1 << "\t\t" << bt[i] <<"\t  " << wt[i] <<"\t\t   " << tat[i] <<endl;
   }
```

```
   cout << "Average waiting time = " << (float)total_wt / (float)n;
   cout << "\nAverage turn around time = " << (float)total_tat / (float)n;
}

int main()
{
   int A,B,C,D,E;
   cout<<"ENTER JOBS"<<endl;
   cin >>A>>B>>C>>D>>E;
   int job[] = { A, B, C, D, E};
   int n = sizeof job / sizeof job[0];
   int B1,B2,B3,B4,B5;
   cout<<"ENTER BURST TIME"<<endl;
   cin>>B1>>B2>>B3>>B4>>B5;
   int burst_time[] = {B1, B2, B3, B4, B5};
   findavgTime(job, n, burst_time);
   return 0;
}
```

CODE for SJF:

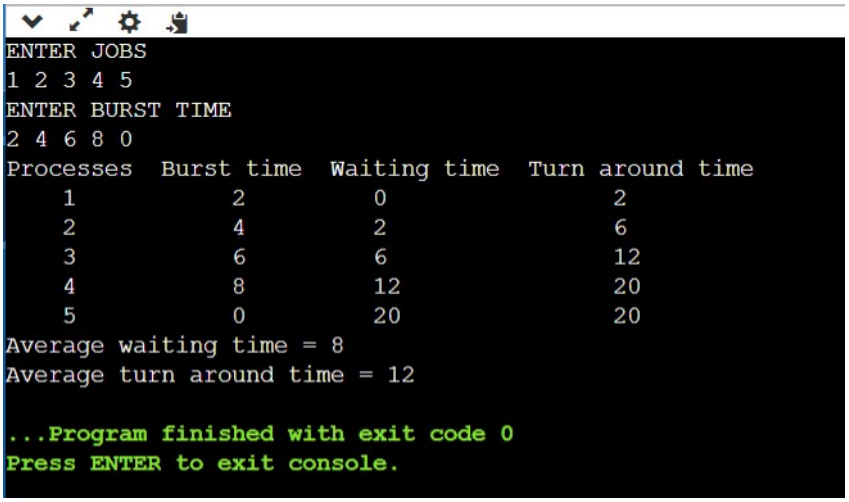
```
#include <iostream>
using namespace std;
int main()
{
       int A[100][4];
       int i, j, n, total = 0, index, temp;
       float avg_wt, avg_tat;
       cout << "Enter number of process: ";
       cin >> n;
       cout << "Enter Burst Time:" << endl;
      for (i = 0; i < n; i++)
       {
          //   cout << "P" << i + 1 << ": ";
            cin >> A[i][1];
            A[i][0] = i + 1;
       }
       for (i = 0; i < n; i++)
      {
            index = i;
            for (j = i + 1; j < n; j++)
            if (A[j][1] < A[index][1])
            index = j;
           temp = A[i][1];
           A[i][1] = A[index][1];
           A[index][1] = temp;
           temp = A[i][0];
           A[i][0] = A[index][0];
           A[index][0] = temp;
     }
     A[0][2] = 0;
      for (i = 1; i < n; i++)
     {
            A[i][2] = 0;
            for (j = 0; j < i; j++)
            A[i][2] += A[j][1];
```

Name:
Reg.no:

```
        total += A[i][2];
    }
    avg_wt = (float)total / n;
    total = 0;
    cout << "Process   BurstTime  WaitingTime  TurnaroundTime" << endl;
    for (i = 0; i < n; i++) {
    A[i][3] = A[i][1] + A[i][2];
    total += A[i][3];
    cout << "   P" << A[i][0] << "         " << A[i][1] << "          " << A[i][2] << "        " << A[i][3] << endl;
    }
    avg_tat = (float)total / n;
    cout << "Average Waiting Time = " << avg_wt << endl;
    cout << "Average Turnaround Time = " << avg_tat << endl; }
```

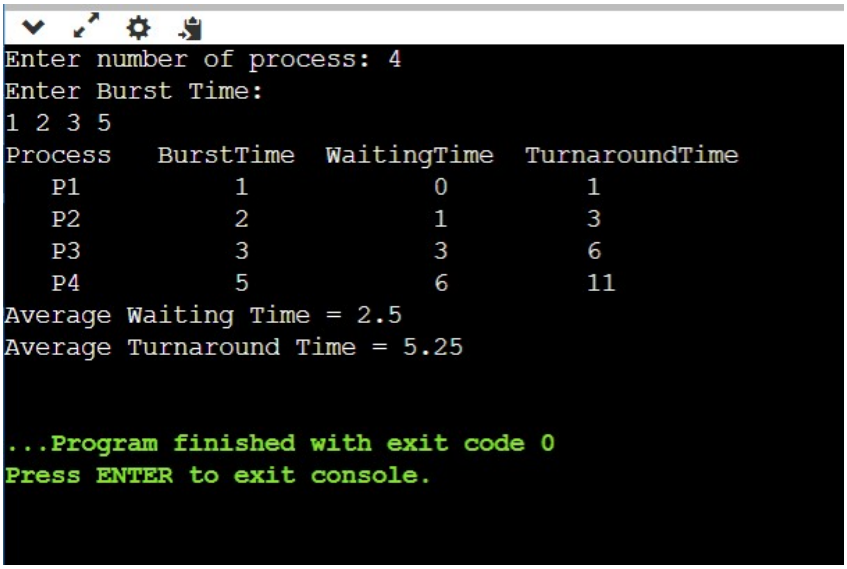
OUTPUT:

FCFS:



```
ENTER JOBS
1 2 3 4 5
ENTER BURST TIME
2 4 6 8 0
Processes  Burst time  Waiting time  Turn around time
    1          2          0               2
    2          4          2               6
    3          6          6               12
    4          8          12              20
    5          0          20              20
Average waiting time = 8
Average turn around time = 12

...Program finished with exit code 0
Press ENTER to exit console.
```

SJF:



```
Enter number of process: 4
Enter Burst Time:
1 2 3 5
Process   BurstTime  WaitingTime  TurnaroundTime
   P1          1           0          1
   P2          2           1          3
   P3          3           3          6
   P4          5           6          11
Average Waiting Time = 2.5
Average Turnaround Time = 5.25


...Program finished with exit code 0
Press ENTER to exit console.
```

RESULT:

The average turnaround time and average waiting time is calculated successfully using FCFS and SJF CPU scheduling algorithm.

Name:
Reg.no:

Ex no: 05
Date: 09.05.2024

Implementation of Priority and Round Robin CPU Scheduling

AIM:

To write a program for the Priority CPU Scheduling and Round Robin.

ALGORITHM:

1. Start the Program.
2. Declare the required inputs.
3. Get the input of burst time, arrival time, priority from the user.
4. Compute the required calculation like
    i. Turnaround Time = Completion Time - Arrival Time
    ii. Waiting Time = Turnaround Time – Burst Time

Step - 5: Print the required output.

Step - 6: Stop the program

PROGRAM:

Code for Priority Scheduling:

```
#include<iostream>
#include<limits>
using namespace std;

class Process {
    public:
        string processName;
        int arrivalTime;
        int burstTime;
        int priority;

        int remainingTime;
        int responseTime;
        int completionTime;
        int waitingTime;
        int turnAroundTime;

        void initialize(){
               remainingTime = burstTime;
        }
};

int main(){
    int numOfProcesses;
```

Name:
Reg.no:

```
cout << "Enter no. of processes: ";
cin >> numOfProcesses;
cout<<numOfProcesses<<endl;

Process processes[numOfProcesses];

for(int n=0;n<numOfProcesses;n++){
    cin >> processes[n].processName;
    cin >> processes[n].arrivalTime;
    cin >> processes[n].burstTime;
    cin >> processes[n].priority;

    processes[n].initialize();
}
cout << "\n" << endl;
for (int i=0; i<numOfProcesses-1; i++) {
    for (int j=i+1; j<numOfProcesses; j++) {
        if(processes[j].arrivalTime < processes[i].arrivalTime) {
            Process temp = processes[j];
            processes[j] = processes[i];
            processes[i] = temp;
        }
    }
}

int currentTime = 0;

while(true) {

    int currentHighestPriorityIndex = -1;
    int currentHighestPriority = numeric_limits<int>::max();

    bool isAllCompleted = true;

    for (int i=0; i<numOfProcesses; i++){
            if(processes[i].remainingTime > 0){
                    isAllCompleted = false;
             if(processes[i].arrivalTime <= currentTime){
                 if(processes[i].priority < currentHighestPriority){
                     currentHighestPriority = processes[i].priority;
                     currentHighestPriorityIndex = i;
                 }
             }

            }
    }

    if(isAllCompleted) {
            break;
    }
```

Name:

Reg.no:

```
                processes[currentHighestPriorityIndex].responseTime = currentTime;
            processes[currentHighestPriorityIndex].remainingTime = 0;
            currentTime += processes[currentHighestPriorityIndex].burstTime;
            processes[currentHighestPriorityIndex].completionTime = currentTime;
        }
        int sumResponseTime = 0;
        int sumCompletionTime = 0;
        int sumWaitingTime = 0;
        int sumTurnAroundTime = 0;

        for(int n=0;n<numOfProcesses;n++){
            processes[n].turnAroundTime = processes[n].completionTime - processes[n].arriv
alTime;
            processes[n].waitingTime = processes[n].turnAroundTime - processes[n].burstTim
e;

            sumResponseTime += processes[n].responseTime;
            sumCompletionTime += processes[n].completionTime;
            sumWaitingTime += processes[n].waitingTime;
            sumTurnAroundTime += processes[n].turnAroundTime;
        }
        cout<<"Process\t"<<"Arrival Time\t"<<"Burst Time\t"<<"Priority\t"<<"Completion Tim
e\t"<<endl;
        for(int i=0;i<numOfProcesses;i++){
            cout<<" "<< processes[i].processName <<"\t"<< processes[i].arrivalTime <<"\t\t
"<< processes[i].burstTime <<"\t\t"<< processes[i].priority<<"\t\t"<<
             processes[i].completionTime <<endl;

        }
        cout<<endl;
        cout<<"TurnAround Time\t"<<"Waiting Time"<<endl;
         for(int i=0;i<numOfProcesses;i++){
            cout<< processes[i].turnAroundTime<<"\t\t"<< processes[i].waitingTime<<endl;
         }

        cout << "\n\nAverage Waiting Time for " << (numOfProcesses) << " Processes: " << (
float) sumWaitingTime/numOfProcesses;
        cout << "\n\nAverage Turn Around Time for " << (numOfProcesses) << " Processes: "
<< (float) sumTurnAroundTime/numOfProcesses;

        return 0;
}
```

Name:
Reg.no:

OUTPUT:

```
Compiled Successfully. memory: 3704 time: 0.23 exit code: 0

Enter no. of processes: 5


Process Arrival Time    Burst Time      Priority        Completion Time
 P1     0               5               1               5
 P2     2               3               3               8
 P3     4               1               5               15
 P4     6               2               2               10
 P5     8               4               4               14

TurnAround Time Waiting Time
5               0
6               3
11              10
4               2
6               2


Average Waiting Time for 5 Processes: 3.4

Average Turn Around Time for 5 Processes: 6.4
```

CODE for Round Robin Scheduling:

```
#include<iostream>
using namespace std;
class Process{
    public:
        string processName;
        int burstTime;
        int arrivalTime;
        int waitingTime;
        int completionTime;
        int responseTime;
        int turnAroundTime;
        int remainingTime;
        void initialize(){
            waitingTime = 0;
            responseTime = 0;
            turnAroundTime = 0;
            remainingTime = burstTime;
        }
};

int main(){
    int numOfProcesses;
    int timeQuantum;
    int currentTime = 0;
    cout << "\nEnter Time Quantum: ";
    cin >> timeQuantum;
    cout<<timeQuantum<<endl;
    cout << "\nEnter no. of processes: ";
```

Name:
Reg.no:

```
    cin >> numOfProcesses;
    cout<<numOfProcesses<<endl;
    Process processes[numOfProcesses];
    for(int n=0;n<numOfProcesses;n++){
        // cout << "Enter Process Name for " << (n+1) << ": ";
        cin >> processes[n].processName;
        // cout << "Enter Arrival Time for Process " << (n+1) << ": ";
        cin >> processes[n].arrivalTime;
        // cout << "Enter Burst Time for Process " << (n+1) << ": ";
        cin >> processes[n].burstTime;
        processes[n].initialize();
    }
    cout << "\n" << endl;
    currentTime = processes[0].arrivalTime;
    int remainingProcesses = numOfProcesses;
    for(int i=0;i<numOfProcesses;i=(i+1)%numOfProcesses){
        if(processes[i].remainingTime > 0 && processes[i].arrivalTime <= currentTime){
            if(processes[i].remainingTime == processes[i].burstTime){
                  processes[i].responseTime = currentTime;
            }
            if(processes[i].remainingTime <= timeQuantum){
                  currentTime += processes[i].remainingTime;
                  processes[i].completionTime = currentTime;
                  processes[i].remainingTime = 0;
                remainingProcesses--;
            }
            else{
                  currentTime += timeQuantum;
                  processes[i].remainingTime -= timeQuantum;
            }
        }
        if(remainingProcesses == 0){
            break;
        }
    }
    int sumResponseTime = 0;
    int sumCompletionTime = 0;
    int sumWaitingTime = 0;
    int sumTurnAroundTime = 0;

    for(int n=0;n<numOfProcesses;n++){
        processes[n].turnAroundTime = processes[n].completionTime -
processes[n].arrivalTime;
        processes[n].waitingTime = processes[n].turnAroundTime -
processes[n].burstTime;
        sumResponseTime += processes[n].responseTime;
        sumCompletionTime += processes[n].completionTime;
        sumWaitingTime += processes[n].waitingTime;
        sumTurnAroundTime += processes[n].turnAroundTime;
    }
    cout<<"Process\t"<<"Arrival Time\t"<<"Burst Time\t"<<"Completion Time\t"<<endl;
```

Name:
Reg.no:

```
    for(int i=0;i<numOfProcesses;i++){
        cout<<" "<< processes[i].processName <<"\t"<< processes[i].arrivalTime
<<"\t\t"<< processes[i].burstTime <<"\t\t"<<
         processes[i].completionTime <<endl;

    }
    cout<<endl;
    cout<<"TurnAround Time\t"<<"Waiting Time"<<endl;
     for(int i=0;i<numOfProcesses;i++){
        cout<< processes[i].turnAroundTime<<"\t\t"<< processes[i].waitingTime<<endl;
     }
    cout << "\n\nAverage Waiting Time for " << (numOfProcesses) << " Processes: " <<
(float) sumWaitingTime/numOfProcesses;
    cout << "\n\nAverage Turn Around Time for " << (numOfProcesses) << " Processes: "
<< (float) sumTurnAroundTime/numOfProcesses;
    return 0;
}
```

OUTPUT:

```
Compiled Successfully. memory: 3804 time: 0.19 exit code: 0


Enter Time Quantum: 5

Enter no. of processes: 6


Process Arrival Time    Burst Time      Completion Time
 P1     0               7               31
 P2     1               4               9
 P3     2               15              55
 P4     3               11              56
 P5     4               20              66
 P6     4               9               50

TurnAround Time Waiting Time
31              24
8               4
53              38
53              42
62              42
46              37


Average Waiting Time for 6 Processes: 31.1667

Average Turn Around Time for 6 Processes: 42.1667
```

RESULT:

The average turnaround time and average waiting time is calculated successfully using Priority and Round Robin CPU scheduling.

Name:
Reg.no:

Ex no: 06
Date: 10.05.2024

Use of process, file, stat and Directory system calls

A.Use of file system calls:

Aim:

To write a program to exibit the concept of file system call and its use.

Algorithm:

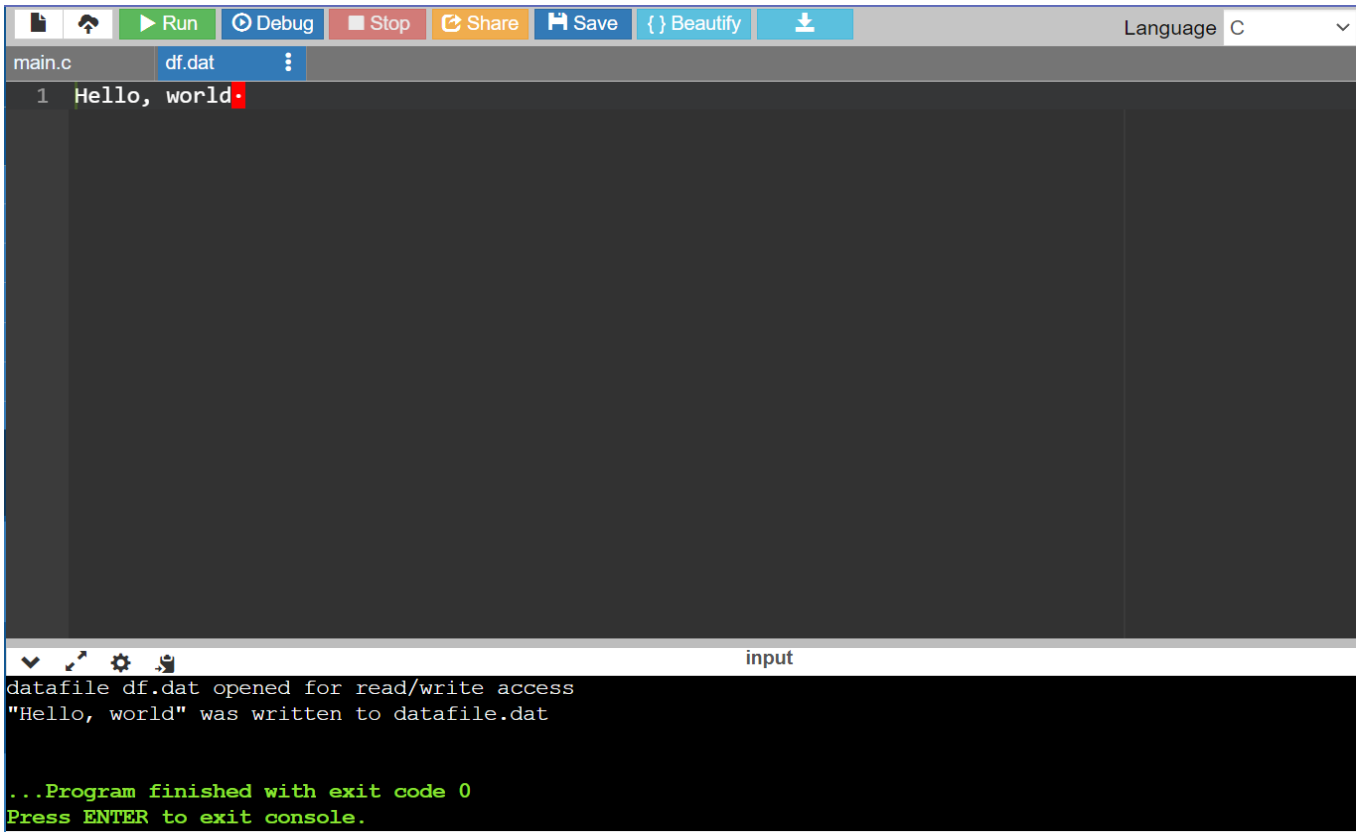
1. Initialize message and buffer.
2. Open the file df.dat.
3. Check if the file opened successfully: If fd is not -1, print "datafile df.dat opened for read/write access".
4. Write the message to the file:Use write(fd, message, sizeof(message))`.
5. Move the file pointer to the beginning: Use lseek(fd, 0L, 0).
6. Read the message from the file into the buffer: Use read(fd, buffer, sizeof(message)).
7. Check if read operation was successful: If the read size equals sizeof(message), print the buffer content. Else, print "*** error reading datafile.dat ***".
8. Close the file.
9. Handle file open failure: If fd is -1, print "*** datafile.dat already exists ***".
10. Exit the program.

PROGRAM:

```
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <stdlib.h>

static char message [] = "Hello, world";

int main() {
	int fd;
	char buffer [80];
	fd = open("df.dat",O_RDWR | O_CREAT | O_EXCL, S_IREAD |S_IWRITE);
	if (fd != -1) {
		printf("datafile df.dat opened for read/write access\n");
		write(fd, message, sizeof(message));
		lseek(fd, 0L, 0);
		if (read(fd, buffer, sizeof(message)) == sizeof (message))
			printf("\"%s\" was written to datafile.dat\n", buffer);
		else
			printf("*** error reading datafile.dat ***\n");
			close (fd);
	}
	else
		printf("*** datafile.dat already exists ***\n");
		exit (0);
	}
}
```

Name:
Reg.no:

OUTPUT:



Name:
Reg.no:

B.Use of Directory system calls

AIM:

To write a program to illustrate the concept of directory system call.

ALGORITHM:

1. Initialize variables.
2. Open the directory.
3. Prompt for the file name:
4. Read directory entries: Use a while loop to iterate over entries with readdir(dir).
5. Compare file names: Within the loop, use strcmp(a, ent->d_name) to compare the entered file name with each directory entry's name. If a match is found, print the file name and set flag to 1, then break the loop.
6. Check if the file was found: After the loop, check if flag is 1.If it is, print "The given file is found". Otherwise, print "File not found".
7. lose the directory: If closedir returns a non-zero value, print an error message.
8. Main function: Declare the directory name. Print a message asking for the directory name to be searched.
9. Call the search function:
10. Return from main: Return 0 to indicate successful execution.

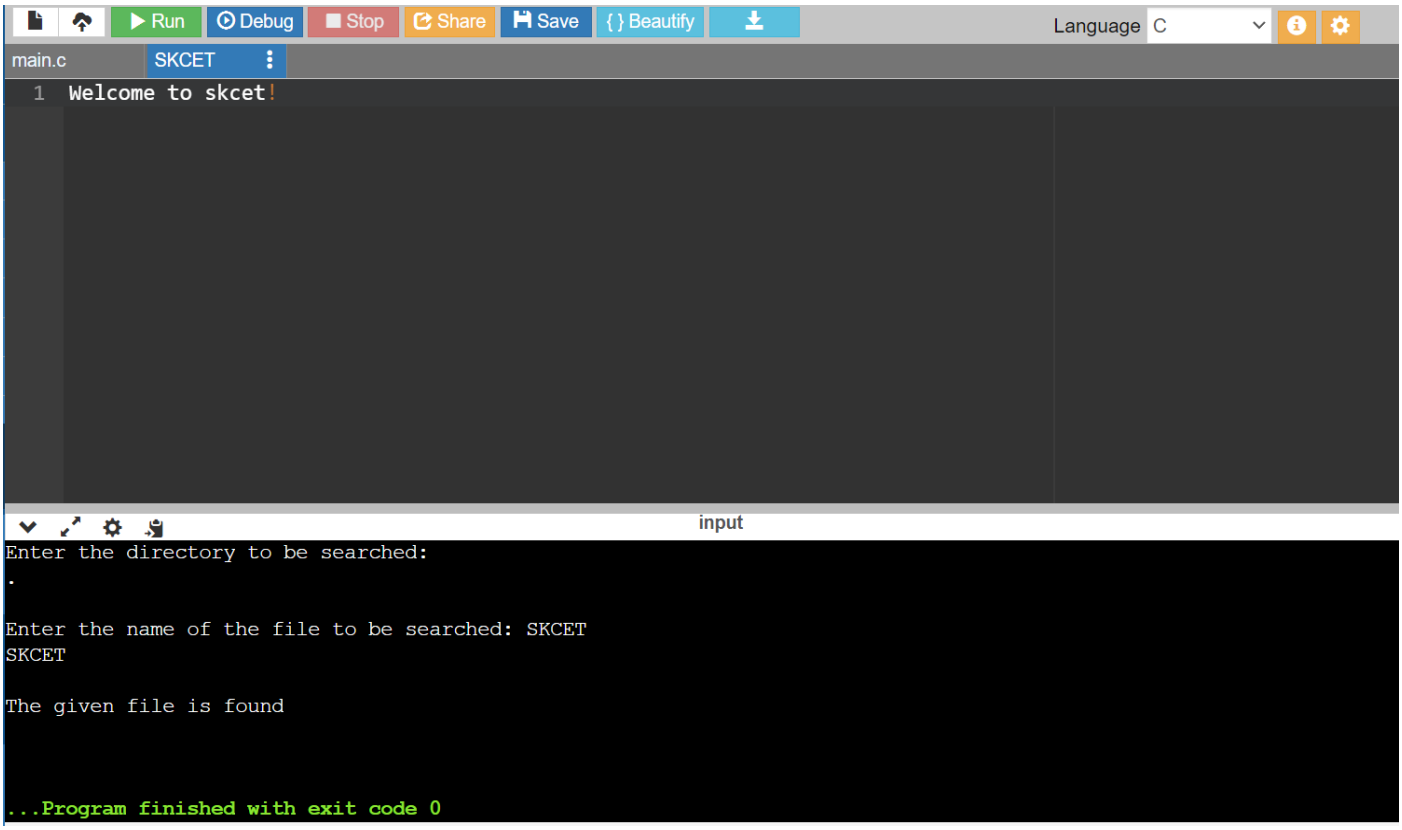
PROGRAM:

```
#include<stdio.h>
#include<dirent.h>
#include<stdlib.h>
#include<string.h>
void sea(char *dname) {
	DIR *dir;
	struct dirent *ent;
	int flag = 0;
	char a[15];
	if ((dir= opendir(dname))==NULL) {
		printf("\n unable to open directory ");
		exit(1);
	}
	printf("\n Enter the name of the file to be searched :");
	scanf("%s",a);
	while((ent=readdir(dir))!=NULL) {
		if(!strcmp(a,ent->d_name)) {
			printf("%s",ent->d_name);
			flag++;
		}
	}
	if(flag==1) printf("\n the given file is found\n\n");
	else printf("\nfile not found");

	if(closedir(dir)!=0) printf("unable to close directory");
}

void main() {
	char dirname[25];
	printf("\n Enter the directory to be searched :\n");
	scanf("%s",dirname);
	sea(dirname);
```

Name:
Reg.no:

```
}
```

OUTPUT:



Name:
Reg.no:

C.Use of stat system calls:

AIM:

To write a program to exhibit the concept of stat system call and its use.

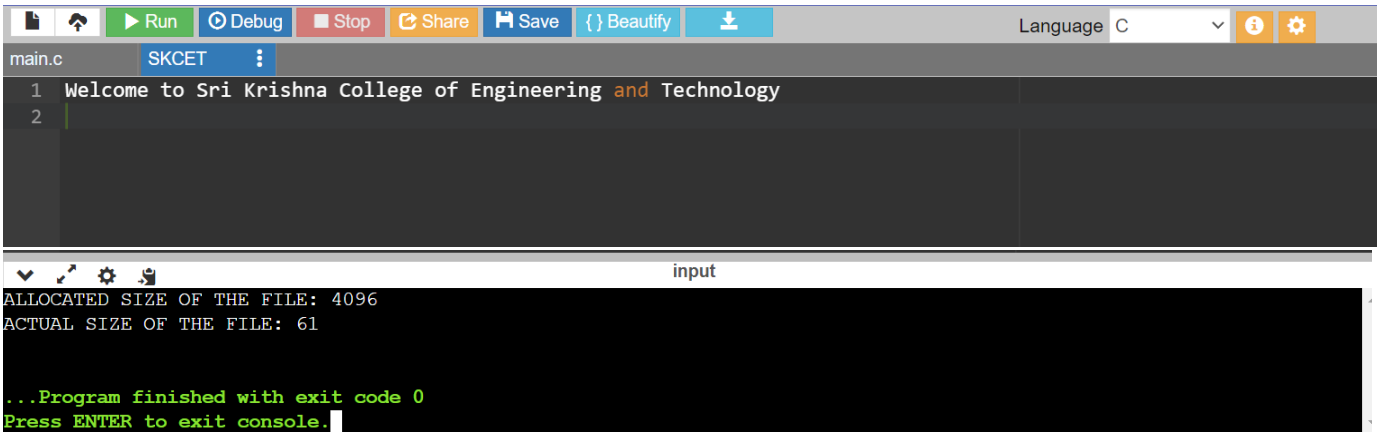
ALGORITHM:

1. Include necessary headers.
2. Declare struct stat s and integers a and b.
3. Call stat to get file statistics.
4. Check if stat returns -1.
5. Print error message and exit if stat fails.
6. Assign s.st_blksize to a.
7. Assign s.st_size to b.
8. Print "ALLOCATED SIZE OF THE FILE" and value of a.
9. Print "ACTUAL SIZE OF THE FILE" and value of b.
10. Return 0 from main. Close the program.

PROGRAM:

```
#include <stdio.h>
#include <sys/stat.h>
#include <stdlib.h>
int main()  {
       struct stat s;
        int a, b;
       if (stat("SKCET", &s) == -1) {
                    perror("Error: cannot stat file");
                    exit(0);
       }
       a = s.st_blksize;
       b = s.st_size;
       printf("ALLOCATED SIZE OF THE FILE: %d\n", a);
       printf("ACTUAL SIZE OF THE FILE: %d\n", b);
        return 0;
}
```

OUTPUT:



Name:
Reg.no:

D.Use of process system calls:

AIM:

To write a program to exhibit the concept of process system calls.

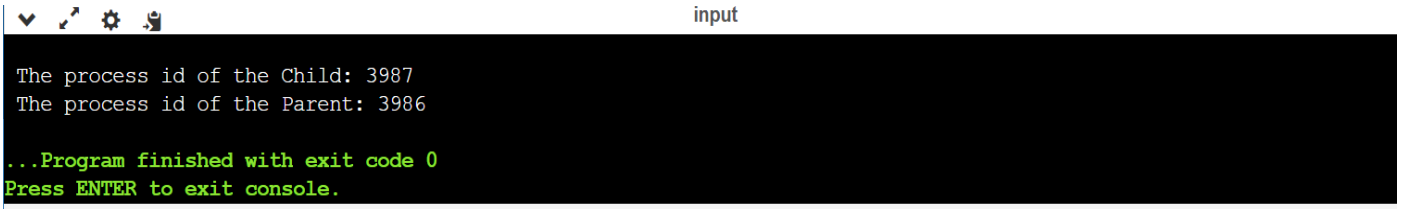
ALGORITHM:

1. Include necessary headers.
2. Declare a struct stat variable.
3. Declare integer variables.
4. Retrieve file statistics: Use stat("SKCET", &s) to get file statistics for the file named "SKCET".
5. Check for errors: If stat returns -1, print an error message using perror and exit the program with exit(0).
6. Extract block size: Assign s.st_blksize to a.
7. Extract file size: Assign s.st_size to b.
8. Print the allocated size of the file (stored in a).
9. Print the actual size of the file (stored in b).
10. Return from main: Return 0 to indicate successful execution.

PROGRAM:

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
int main() {
	int pid = fork(); // Create a new process
	if (pid < 0) { // Check if fork failed
		printf("The fork cannot be created\n");
		exit(0);
	}
	else if (pid == 0) { // In child process
		execlp("/bin/ls", "ls", NULL); // Execute command
		printf("\n The process id of the Child : %d", getpid());
		printf("\n The process id of the Parent : %d", getppid());
	}
	else { // In parent process
		printf("\n The process id of the Child: %d", getpid());
		printf("\n The process id of the Parent: %d", getppid());
	}
	return 0; // Return from main
}
```

OUTPUT:



RESULT:

The above program has been executed and the output has been shown successfully.

Name:
Reg.no:

Ex no: 07
Date: 11.05.2024

C Simulation of vi, cp and cat commands

AIM:

To write a c program for simulatión of vi, cat and Cp commands.

ALGORITHM:

1. Start the program and initialize Variables
2. Display Menu and get User Choice
3. Switch on User Choice
4. Case 1 - vi Command
   a. Print Termination Instruction
   b. Open File for Writing (vi Command)
   c. Read and Write Characters to File
5. Case 2 - cat Command
   a. Open File for Reading (cat Command)
   b. Read and Display File Content
6. Case 3 - cp Command
   a. Open Source and Destination Files (cp Command), Copy File Content
7. End the program

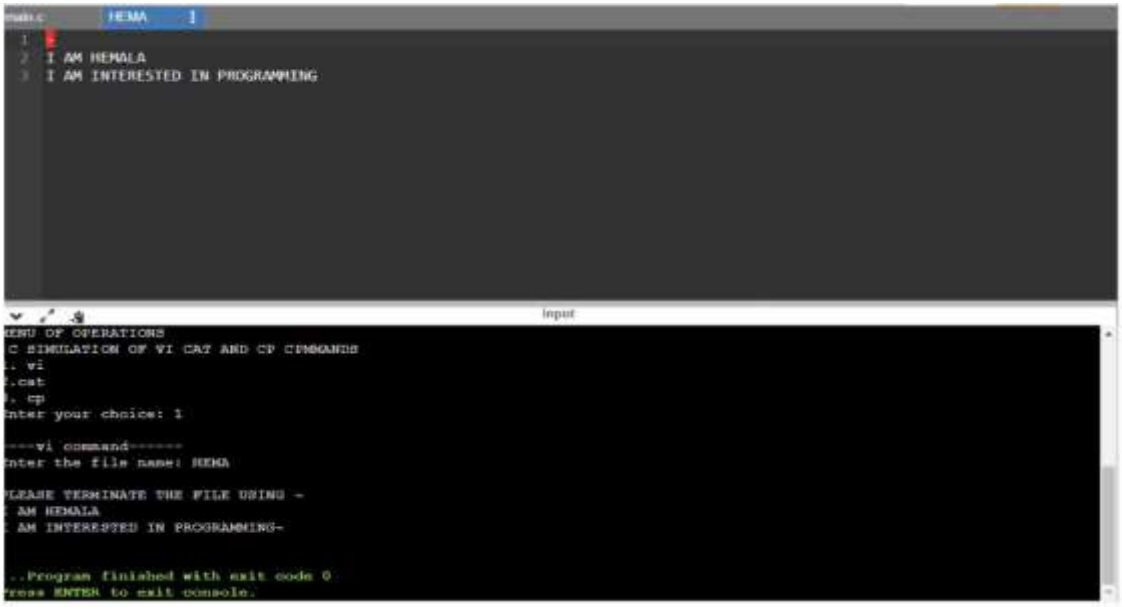
PROGRAM:

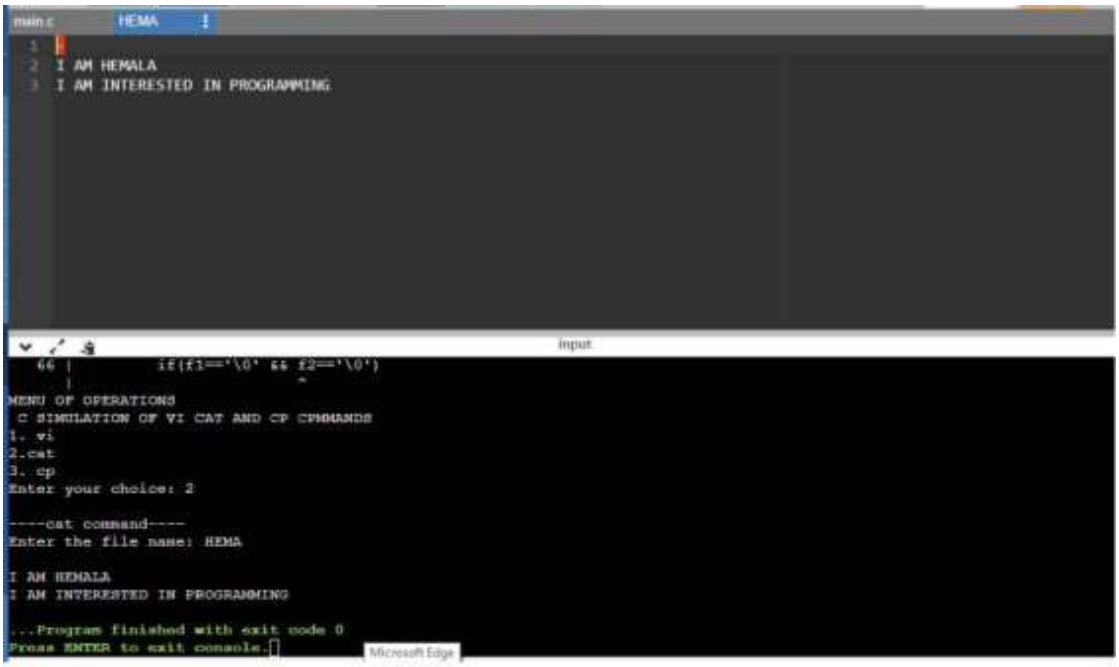
```
#include<stdio.h>
#include<stdlib.h>

int main() {
    int ch;
    char a,b,fn1[10],fn2[10];

    FILE *f1,*f2;

    printf("MENU OF OPERATIONS\n C SIMULATION OF VI CAT AND CP COMMANDS ");

    printf("\n1. vi \n2.cat \n3. cp ");
    printf("\nEnter your choice: ");
    scanf("%d",&ch);
    switch(ch) {
        case 1:
             printf("\n----vi command ");
             printf("\nEnter the file name: ");
             scanf("%s",fn1);
             printf("\n PLEASE TERMINATE THE FILE USING ~ ");
             printf("\n");

             f1=fopen(fn1,"w");
             while(a!='~') {
                   fputc(a,f1); a=getchar();
             }
             fclose(f1);
             break;
```

```
      case 2:
            printf("\n----cat command      ");
            printf("\n Enter the file name: ");
            scanf("%s",fn1);
            f1=fopen(fn1,"r");

            if(f1=='\0') {
                  printf("\n File is empty"); exit(0);
            } else {
                  a=fgetc(f1);
                  while(a!=EOF) {
                        printf("%c",a); a=fgetc(f1);
                  }
            }

            fclose(f1);
            break;
      case 3:
            printf("\n----cp command ");

            printf("\nEnter the source file name: "); scanf("%s",fn1);
            printf("\nEnter the destination file name: "); scanf("%s",fn2);
            f1=fopen(fn1,"r");

            f2=fopen(fn2,"w"); if(f1=='\0' && f2=='\0') {
                  printf("\nFile is empty");
            } else {
                  b=fgetc(f1);
                  while(b!=EOF) {
                        fputc(b,f2);
                        b=getc(f1);
                  }
            }

            fclose(f1);
            fclose(f2);

            printf("\n File is copied successfully");
            break;
      default:
            printf("\nEnter a valid option");
    }
}
```

Name:
Reg.no:

OUTPUT:





RESULT:

Thus the above program to simulate the vi, cat and cp commands was compiled andexecuted successfully.

Name:
Reg.no:

Ex no: 08
Date: 13.05.2024

Simulation of Producer Consumer Problem

AIM:

To simulate the Producer-Consumer problem using C programming language, illustrating the synchronization between producer and consumer threads using mutexes and condition variables.

ALGORITHM:

1. **Initialize:**
   - Create a buffer with a fixed size.
   - Initialize mutex and condition variables for synchronization.
   - Create producer and consumer threads.

2. **Producer:**
   - Loop to produce an item.
   - Acquire the mutex lock.
   - Check if the buffer is full; if full, wait for the consumer to consume.
   - Add the item to the buffer.
   - Signal the consumer that an item is available.
   - Release the mutex lock.
   - Sleep for a random amount of time.

3. **Consumer:**
   - Loop to consume an item.
   - Acquire the mutex lock.
   - Check if the buffer is empty; if empty, wait for the producer to produce.
   - Remove the item from the buffer.
   - Signal the producer that space is available.
   - Release the mutex lock.
   - Sleep for a random amount of time.

4. **Main Function:**
   - Create and start producer and consumer threads.
   - Join the threads to wait for their completion.

PROGRAM:

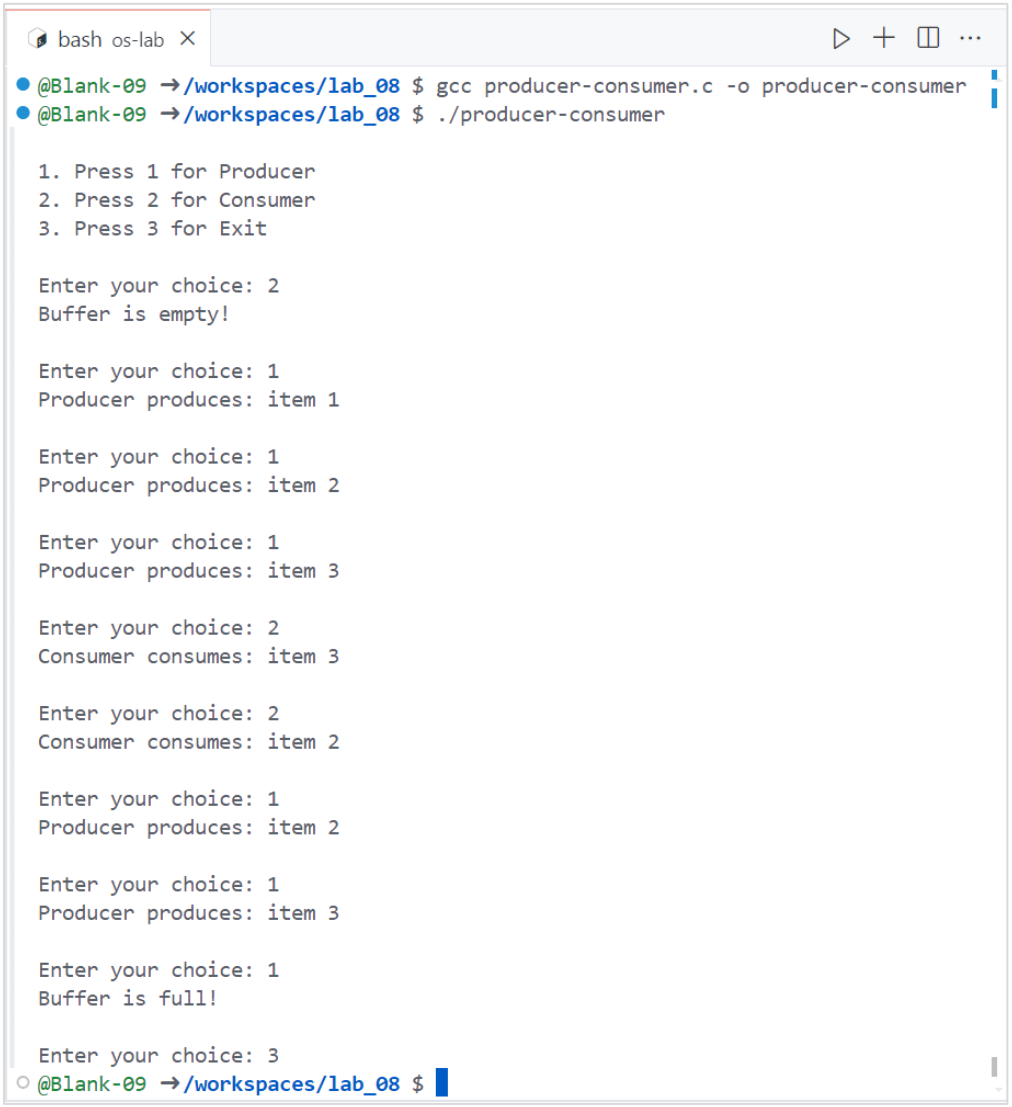
```
#include <stdio.h>
#include <stdlib.h>

int mutex = 1;
int full = 0;
int empty = 3, x = 0;

void producer() {
  --mutex;
  ++full;
  --empty;
  x++;

  printf("Producer produces: "
```

Name:
Reg.no:

```
        "item %d",
        x);

  ++mutex;
}

void consumer() {
  --mutex;
  --full;
  ++empty;

  printf("Consumer consumes: "
        "item %d",
        x);

  x--;
  ++mutex;
}

int main() {
  int n, i;
  printf("\n1. Press 1 for Producer"
        "\n2. Press 2 for Consumer"
        "\n3. Press 3 for Exit");

#pragma omp critical

  for (i = 1; i > 0; i++) {

    printf("\n\nEnter your choice: ");
    scanf("%d", &n);

    switch (n) {
    case 1:
      if ((mutex == 1) && (empty != 0)) {
        producer();
      } else {
        printf("Buffer is full!");
      }
      break;

    case 2:
      if ((mutex == 1) && (full != 0)) {
        consumer();
      } else {
        printf("Buffer is empty!");
      }
      break;

    case 3:
      exit(0);
      break;
    }
  }
}
```

Name:
Reg.no:

OUTPUT:



RESULT:

The simulation of Producer Consumer problem program is successfully compiled and executed. And the output is verified by the sample output.

Name:
Reg.no:

Ex no: 09
Date: 14.05.2024

Banker's Algorithm

AIM:

To identify the safe state execution without any deadlocks using banker's algorithm in c++;

ALGORITHM:

1) Initialize Data Structures
    i) Input the number of processes (P) and resources (R).
    ii) Define and initialize the Allocation, Maximum, Available, Need, Finished arrays, and the Safe Sequence.
2) Input Allocation and Maximum Matrices
    i) For each process, input the allocated resources.
    ii) For each process, input the maximum resources required.
    iii) Input Available Resources
    iv) Input the initial available resources.
3) Calculate Need Matrix
    i) For each process, calculate the Need matrix as Need[i][j] = Maximum[i][j] - Allocation[i][j].
4) Display the Allocation, Maximum, and Need matrices along with Available resources for verification.
5) Check Safe State
    i) Initialize all processes as not finished (Finished[i] = 0).
    ii) Initialize the safe sequence index to 0.
6) Find Safe Sequence
    i) For each process, if it is not finished, check if its needs can be met with the current available resources.
    ii) If yes, add the process to the safe sequence, mark it as finished, and release its resources back to the available pool.
    iii) Repeat until all processes are checked.
7) Verify Safe State
    i) If all processes are finished, the system is in a safe state. Otherwise, it is not.
8) Output Results
    i) If the system is in a safe state, output the safe sequence.
    ii) If not, indicate that the system is not in a safe state.

PROGRAM:

```
#include <iostream>
using namespace std;

int main() {
    cout << "-----------BANKER'S ALGORITHM-----------\n\n";
    int process, resources, index = 0;
    cout << "Enter the number of process(es): ";
    cin >> process;
    cout << "Enter the number of resource(s): ";
    cin >> resources;
    int allocation[process][resources];
    int maximum[process][resources];
    int available_resource[resources];
    for (int i = 0; i < process; i++) {
        cout << "\nEnter the allocations for process P" << i << ": ";
        for (int j = 0; j < resources; j++) {
            cin >> allocation[i][j];
```

Name:
Reg.no:

```
    }
    cout << "Enter the maximum resource for process P" << i << ": ";
    for (int j = 0; j < resources; j++) {
        cin >> maximum[i][j];
    }
}
cout << "\n\nEnter the initial available resources: ";
for (int i = 0; i < resources; i++) {
    cin >> available_resource[i];
}
cout << "\nProcess Table: ";
cout << "\n---------------------------------";
cout << "\nProcess | Allocations | Maximum |";
cout << "\n---------------------------------";
for (int i = 0; i < process; i++) {
    cout << "\nP" << i << "      | ";
    for (int j = 0; j < resources; j++) {
        cout << allocation[i][j] << " ";
    }
    cout << "      | ";
    for (int j = 0; j < resources; j++) {
        cout << maximum[i][j] << " ";
    }
    cout << "  | ";
}
cout << "\n---------------------------------";
cout << "\n\nAvailable Resources: ";
for (int i = 0; i < resources; i++) {
    cout << available_resource[i] << " ";
}
cout << "\n\nNeed Matrx: ";
cout << "\n-------------------";
cout << "\nProcess | Need    |";
cout << "\n-------------------";
int need[process][resources];
for (int i = 0; i < process; i++) {
    cout << "\nP" << i << "      | ";
    for (int j = 0; j < resources; j++) {
        need[i][j] = maximum[i][j] - allocation[i][j];
        cout << need[i][j] << " ";
    }
    cout << "  |";
}
cout << "\n-------------------\n";
int safe_sequence[process];
int finished[process];
for (int i = 0; i < process; i++) {
    safe_sequence[i] = 0;
    finished[i] = 0;
}
for (int k = 0; k < process; k++) {
    for (int i = 0; i < process; i++) {
        if (finished[i] == 0) {
            bool flag = true;
```

Name:
Reg.no:

```
                for (int j = 0; j < resources; j++) {
                    if (need[i][j] > available_resource[j]) {
                        flag = false;
                        break;
                    }
                }
                if (flag){
                   safe_sequence[index++] = i;
                   cout<<"\nUpdated Available Resources after allocated to process P"<<i;
                   cout<< ": ";
                   for (int j = 0; j < resources; j++) {
                        available_resource[j] += allocation[i][j];
                        cout << available_resource[j] << " ";
                    }
                    finished[i] = 1;
                }
            }
        }
    }
    bool flag = true;
    for (int i = 0; i < process; i++) {
        if (finished[i] == 0) {
            flag = false;
            cout << "\nThe System is'nt in safe state\n";
            break;
        }
    }
    if (flag) {
        cout << "\n\nThe System is in safe state.";
        cout << "\nSafe Sequence: ";
        for (int i = 0; i < process - 1; i++) {
            cout << "P" << safe_sequence[i] << " -> ";
        }
        cout << "P" << safe_sequence[process - 1];
    }
    cout << "\n\n";
}
```

OUTPUT :

```
----------BANKER'S ALGORITHM----------

Enter the number of process(es): 4
Enter the number of resource(s): 3

Enter the allocations for process P0: 2 1 0
Enter the maximum resource for process P0: 8 6 3

Enter the allocations for process P1: 1 2 2
Enter the maximum resource for process P1: 9 4 3

Enter the allocations for process P2: 0 2 0
Enter the maximum resource for process P2: 5 3 3

Enter the allocations for process P3: 3 0 1
Enter the maximum resource for process P3: 4 2 3
```

Name:
Reg.no:

```
Enter the initial available resources: 4 3 2

Process Table:
--------------------------------
Process | Allocations | Maximum |
--------------------------------
P0      | 2 1 0       | 8 6 3   |
P1      | 1 2 2       | 9 4 3   |
P2      | 0 2 0       | 5 3 3   |
P3      | 3 0 1       | 4 2 3   |
--------------------------------

Available Resources: 4 3 2

Need Matrx:
------------------
Process | Need    |
------------------
P0      | 6 5 3   |
P1      | 8 2 1   |
P2      | 5 1 3   |
P3      | 1 2 2   |
------------------

Updated Available Resources after allocated to process P3: 7 3 3
Updated Available Resources after allocated to process P2: 7 5 3
Updated Available Resources after allocated to process P0: 9 6 3
Updated Available Resources after allocated to process P1: 10 8 5

The System is in safe state.
Safe Sequence: P3 -> P2 -> P0 -> P1
```

RESULT:

The program that demonstrating banker's algorithm was executed and verified successfully.

Name:
Reg.no:

Ex no: 10

Date: 15.05.2024

Dynamic Allocation Strategies

AIM:

To implement and compare three dynamic memory allocation strategies—First Fit, Best-Fit, and Worst-Fit

ALGORITHM:

1. Initialize Memory:
   - Start with a single large free memory block.
2. Allocate Memory:
   - Traverse the list of free memory blocks.
   - Depending on the chosen strategy:
     - First-Fit: Select the first block that is large enough.
     - Best-Fit: Select the smallest block that is large enough.
     - Worst-Fit: Select the largest block that is large enough.
   - If a suitable block is found:
     - Allocate the requested memory.
     - Adjust the size and starting address of the block.
     - If the block size becomes zero, remove it from the free list.
   - If no suitable block is found, indicate that the allocation failed.
3. Deallocate Memory:
   - For each deallocation request:
     - Create a new free block with the specified size and starting address.
     - Insert the new block into the free list.
     - Merge adjacent free blocks, if possible, to reduce fragmentation.
4. Display Free Memory:
   - Traverse the list of free memory blocks.
   - Print the starting address and size of each free block.

PROGRAM:

FIRST FIT:

```
#include <iostream>
#include <vector>
#define MAX 25
using namespace std;
int main() {
   int frag[MAX], b[MAX], f[MAX], i, j, nb, nf, temp, highest = 0;
   static int bf[MAX], ff[MAX];
   int flag, flagn[MAX], fragi = 0, fragx = 0;
   cout << "\n__First Fit__\n";
   cout << "\nEnter the number of blocks: ";
   cin >> nb;
   cout << "Enter the number of processes: ";
   cin >> nf;
   cout << "\nEnter the size of the blocks:-\n";
   for (i = 1; i <= nb; i++) {
```

```
    cout << "Block " << i << ": ";
    cin >> b[i];
    ff[i] = i;
  }
  cout << "Enter the size of the processes:-\n";
  for (i = 1; i <= nf; i++) {
    cout << "Process " << i << ": ";
    cin >> f[i];
  }
  int x = 1;
  cout << "\n\nProcess_No\tProcess_Size\tBlock_No\tBlock_Size\tFragment\n";
  for (i = 1; i <= nf; i++) {
    flag = 1;
    for (j = x; j <= nb; j++) {
      if (f[i] <= b[j]) {
        flagn[j] = 1;
        cout << i << "\t\t" << f[i] << "\t\t" << ff[j] << "\t\t" << b[j] << "\t\t";
        b[j] -= f[i];
        fragi += b[j];
        cout << b[j] << "\n";
        break;
      } else {
        flagn[j] = 0;
        x = 1;
        flag++;
      }
    }
    if (flag > nb) {
      cout << i << "\t\t" << f[i] << "\t\t" << "Has to wait..." << "\t" << "..." << "\t" << "...\n";
    }
  }
  return 0;
}
```

BEST FIT:

```
#include <iostream>
#include <vector>
#define MAX 25
using namespace std;
int main() {
  int frag[MAX], b[MAX], f[MAX], i, j, nb, nf, temp, lowest = 10000;
  static int bf[MAX], ff[MAX], fragi = 0;
  cout << "\n__Best Fit__\n";
  cout << "\nEnter the number of blocks: ";
  cin >> nb;
  cout << "Enter the number of files: ";
  cin >> nf;
  cout << "\nEnter the size of the blocks:-\n";
  for (i = 1; i <= nb; i++) {
    cout << "Block " << i << ": ";
```

```
    cin >> b[i];
    ff[i] = i;
  }
  cout << "Enter the size of the processes :-\n";
  for (i = 1; i <= nf; i++) {
    cout << "Process " << i << ": ";
    cin >> f[i];
  }
  int y, m, z, temp1, flag;
  for (y = 1; y <= nb; y++) {
    for (z = y; z <= nb; z++) {
      if (b[y] > b[z]) {
        temp = b[y];
        b[y] = b[z];
        b[z] = temp;
        temp1 = ff[y];
        ff[y] = ff[z];
        ff[z] = temp1;
      }
    }
  }
  int flagn[MAX];
  int fragx = 0;
  cout << "\n\nProcess_No\tProcess_Size\tBlock_No\tBlock_Size\tFragment\n";
  for (i = 1; i <= nf; i++) {
    flag = 1;
    for (j = 1; j <= nb; j++) {
      if (f[i] <= b[j]) {
        flagn[j] = 1;
        cout << i << "\t\t" << f[i] << "\t\t" << ff[j] << "\t\t" << b[j] << "\t\t";
        b[j] -= f[i];
        fragi += b[j];
        cout << b[j] << "\n";
        break;
      } else {
        flagn[j] = 0;
        flag++;
      }
    }
    if (flag > nb) {
      cout << i << "\t\t" << f[i] << "\t\t" << "Has to wait.." << "\t" << "..." << "\t" << "...\n";
    }
  }
  return 0;
}
```

WORST FIT:

```
#include <iostream>
#define MAX 25
using namespace std;
int main() {
  int frag[MAX], b[MAX], f[MAX], i, j, nb, nf, temp, highest = 0;
  static int bf[MAX], ff[MAX];
  int flag, fragi = 0;
  cout << "\n__Worst Fit__\n";
```

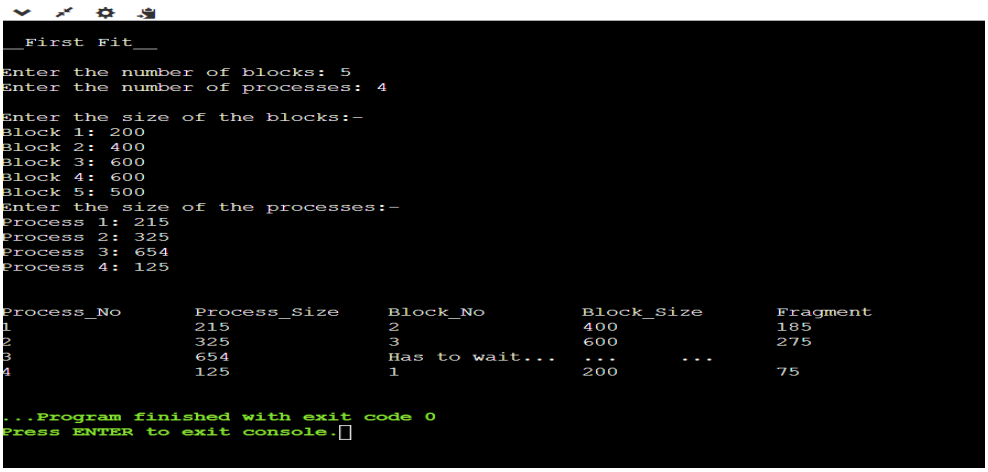
Name:
Reg.no:

```
cout << "\nEnter the number of memory blocks: ";
cin >> nb;
cout << "Enter the number of processes: ";
cin >> nf;
cout << "\nEnter the size of the memory blocks:\n";
for (i = 1; i <= nb; i++) {
   cout << "Block " << i << ": ";
   cin >> b[i];
   ff[i] = i;
}
cout << "Enter the size of the processes:\n";
for (i = 1; i <= nf; i++) {
   cout << "Process " << i << ": ";
   cin >> f[i];
}
 int y, z, temp1;
for (y = 1; y <= nb; y++) {
   for (z = y; z <= nb; z++) {
      if (b[y] < b[z]) {
         temp = b[y];
         b[y] = b[z];
         b[z] = temp;
         temp1 = ff[y];
         ff[y] = ff[z];
         ff[z] = temp1
       }
   }
}
int flagn[MAX];
int fragx = 0;
cout << "\n\nProcess No\tProcess Size\tBlock No\tBlock Size\tRemaining\n";
for (i = 1; i <= nf; i++) {
   flag = 1;
   for (j = 1; j <= nb; j++) {
      if (f[i] <= b[j]) {
         flagn[j] = 1;
         cout << i << "\t\t" << f[i] << "\t\t" << ff[j] << "\t\t" << b[j] << "\t\t";
         b[j] -= f[i];
         fragi += b[j];
         cout << b[j] << "\n";
         break;
      } else {
         flagn[j] = 0;
         flag++;
      }
   }
   if (flag > nb) {
      cout << i << "\t\t" << f[i] << "\t\t" << "Has to wait.." << "\t" << ".." << "\t" << "..\n";
   }
```

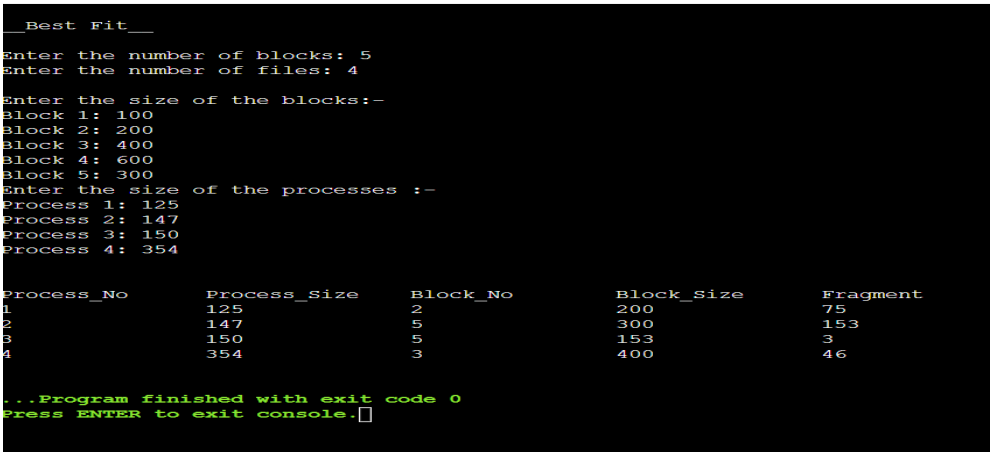
Name:
Reg.no:

```
    }
    return 0;
}
```

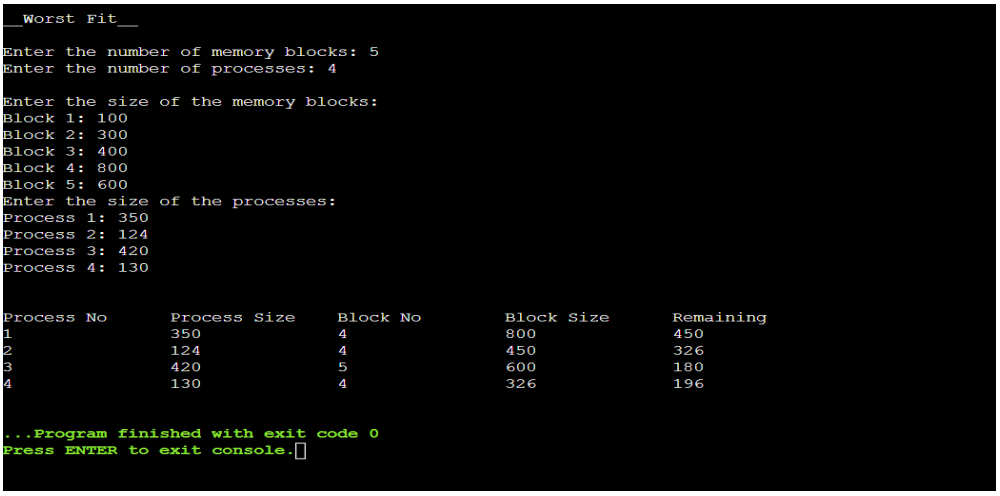
OUTPUT:



```
__First Fit__

Enter the number of blocks: 5
Enter the number of processes: 4

Enter the size of the blocks:-
Block 1: 200
Block 2: 400
Block 3: 600
Block 4: 600
Block 5: 500
Enter the size of the processes:-
Process 1: 215
Process 2: 325
Process 3: 654
Process 4: 125

Process_No      Process_Size    Block_No        Block_Size      Fragment
1               215             2               400             185
2               325             3               600             275
3               654             Has to wait...  ...     ...
4               125             1               200             75

...Program finished with exit code 0
Press ENTER to exit console.
```



```
__Best Fit__

Enter the number of blocks: 5
Enter the number of files: 4

Enter the size of the blocks:-
Block 1: 100
Block 2: 200
Block 3: 400
Block 4: 600
Block 5: 300
Enter the size of the processes :-
Process 1: 125
Process 2: 147
Process 3: 150
Process 4: 354

Process_No      Process_Size    Block_No        Block_Size      Fragment
1               125             2               200             75
2               147             5               300             153
3               150             5               153             3
4               354             3               400             46

...Program finished with exit code 0
Press ENTER to exit console.
```



```
__Worst Fit__

Enter the number of memory blocks: 5
Enter the number of processes: 4

Enter the size of the memory blocks:
Block 1: 100
Block 2: 300
Block 3: 400
Block 4: 800
Block 5: 600
Enter the size of the processes:
Process 1: 350
Process 2: 124
Process 3: 420
Process 4: 130

Process No      Process Size    Block No        Block Size      Remaining
1               350             4               800             450
2               124             4               450             326
3               420             5               600             180
4               130             4               326             196

...Program finished with exit code 0
Press ENTER to exit console.
```

RESULT:

All three dynamic memory allocation strategies—First Fit, Best Fit, and Worst Fit—were implemented and executed successfully.

Name:
Reg.no:

Ex no: 11
Date: 18.05.2024

FIFO and Optimal page replacement algorithm

AIM:

To illustrate FIFO and optimal page replacement algorithm using C++.

ALGORITHM:

1. Start
2. Declare variables
3. Input page numbers and number of frames from the user
4. Initialize variables
5. Create an empty queue to hold pages
6. Loop through each page in the input:
   i. If the page is not already in memory:
      a. If available_frames > 0:
         - Insert the page into the queue and memory
         - Decrease available_frames by 1
      b. If no available frames:
         - Remove the first page from the queue (FIFO)
         - Replace it with the new page
         - Increase page_faults by 1
   ii. If the page is already in memory, continue to the next page
7. Display the total number of page faults
8. Stop

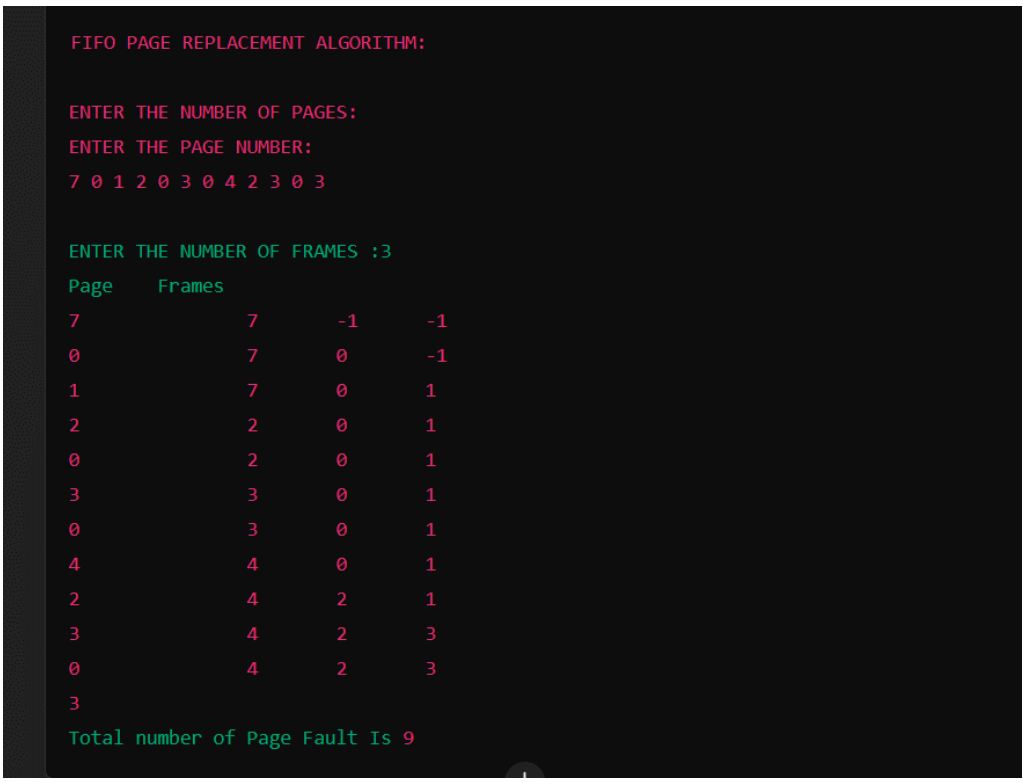
CODE:

```
#include<iostream>
using namespace std;

int main() {
    int i, j, n, a[50], frame[10], no, k, avail, count = 0;
    cout << "FIFO PAGE REPLACEMENT ALGORITHM:\n\n";
    cout << "ENTER THE NUMBER OF PAGES:\n";

    cin >> n;
    cout << "\nENTER THE PAGE NUMBER:\n";

    for (i = 1; i <= n; i++)
        cin >> a[i];

    cout << "\nENTER THE NUMBER OF FRAMES :";
    cin >> no;

    for (i = 0; i < no; i++)
        frame[i] = -1;

    j = 0;

    cout << "Page\tFrames\n";
    for (i = 1; i <= n; i++) {
        cout << a[i] << "\t\t";
        avail = 0;

        for (k = 0; k < no; k++)
```

Name:
Reg.no:

```
            if (frame[k] == a[i])
                avail = 1;

        if (avail == 0) {
            frame[j] = a[i];
            j = (j + 1) % no;
            count++;
            for (k = 0; k < no; k++)
                cout << frame[k] << "\t";
        }


       cout << endl;
 }

  cout << "\nTotal number of Page Fault Is " << count;
  return 0;
}
```

OUTPUT:



Name:
Reg.no:

ALGORITHM:

OPTIMAL:

1.Start.

2.Declare variable.

3.Get an input.

4.If a refused is already present increment count.

5.If not available, Find a page that will not use longer period and Then replace the page with new page.

6.Stop.

CODE:

```
#include <iostream>
using namespace std;

int main() {
    int no_of_frames, no_of_pages, frames[10], pages[30], temp[10];
    int flag1, flag2, flag3, i, j, k, pos, max, faults = 0;

    cout << "___OPTIMAL PAGE REPLACEMENT ALGORITHM:___\n";
    cout << "Enter number of frames: ";
    cin >> no_of_frames;

    cout << "Enter number of pages: ";
    cin >> no_of_pages;

    cout << "Enter page reference string: ";
    for(i = 0; i < no_of_pages; ++i) {
        cin >> pages[i];
    }

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

    for(i = 0; i < no_of_pages; ++i) {
        flag1 = flag2 = 0;
        for(j = 0; j < no_of_frames; ++j) {
            if(frames[j] == pages[i]) {
                flag1 = flag2 = 1;
                break;
            }
        }

        if(flag1 == 0) {
            for(j = 0; j < no_of_frames; ++j) {
                if(frames[j] == -1) {
                    faults++;
                    frames[j] = pages[i];
                    flag2 = 1;
                    break;
                }
            }
        }

        if(flag2 == 0) {
            flag3 = 0;
            for(j = 0; j < no_of_frames; ++j) {
                temp[j] = -1;
                for(k = i + 1; k < no_of_pages; ++k) {
                    if(frames[j] == pages[k]) {
                        temp[j] = k;
```

Name:
Reg.no:

```
                        break;
                    }
                }
            }

            for(j = 0; j < no_of_frames; ++j) {
                if(temp[j] == -1) {
                    pos = j;
                    flag3 = 1;
                    break;
                }
            }

            if(flag3 == 0) {
                max = temp[0];
                pos = 0;
                for(j = 1; j < no_of_frames; ++j) {
                    if(temp[j] > max) {
                        max = temp[j];
                        pos = j;
                    }
                }
            }

            frames[pos] = pages[i];
            faults++;
        }

        cout << "\n";
        for(j = 0; j < no_of_frames; ++j) {
            cout << frames[j] << "\t";
        }
    }

    cout << "\n\nTotal Page Faults = " << faults;
    return 0;
}
```

OUTPUT :

```
___OPTIMAL PAGE REPLACEMENT ALGORITHM:___
Enter number of frames: 3
Enter number of pages: 12
Enter page reference string: 7 0 1 2 0 3 0 4 2 3 0 3

7   -1  -1
7   0   -1
7   0   1
2   0   1
2   0   1
2   0   3
2   0   3
4   0   3
4   2   3
4   2   3
4   2   0
4   2   3

Total Page Faults = 9
```

Result:

The above code is verified and output came successfully.

Name:
Reg.no:

Ex no: 12
Date: 20.05.2024

FCFS Disk Scheduling and SSTF disk scheduling

(a) FCFS:

AIM:

To implement the First-Come, First-Served (FCFS) disk scheduling algorithm in C language, which is used to schedule disk access requests in the order they arrive.

ALGORITHM:

1. **Initialize:**
   - Start with the initial position of the disk head.
   - Create an array to store the sequence of requests.
2. **Calculate Total Movement:**
   - Traverse through the request sequence in the order they arrived.
   - Calculate the total head movement by summing the absolute difference between the current position and the next request in the sequence.
3. **Display the Result:**
   - Print the sequence of requests and the total head movement.
4. End Program.

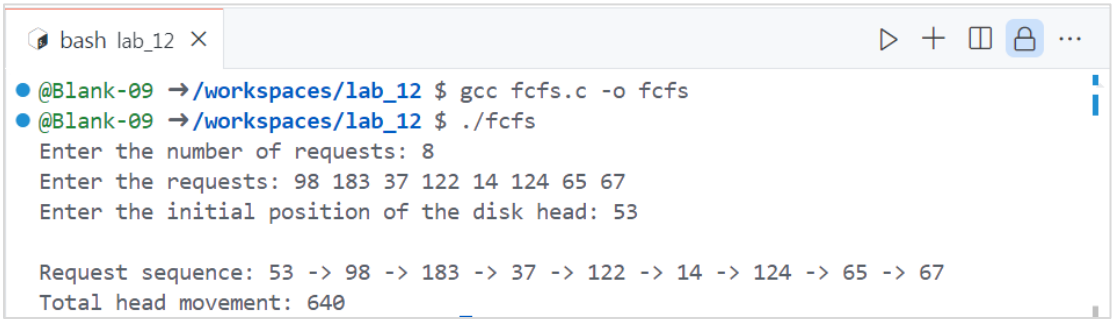
PROGRAM:

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

// Function to calculate absolute difference
int absoluteDifference(int a, int b) {
    return abs(a - b);
}

void FCFS(int requests[], int n, int head) {
    int totalMovement = 0;
    int currentPosition = head;

    printf("Request sequence: %d", head);

    for (int i = 0; i < n; i++) {
        int nextRequest = requests[i];
        totalMovement += absoluteDifference(currentPosition, nextRequest);
        currentPosition = nextRequest;
        printf(" -> %d", nextRequest);
    }

    printf("\nTotal head movement: %d\n", totalMovement);
}

int main() {
    int n, head;

    printf("Enter the number of requests: ");
    scanf("%d", &n);

    int requests[n];
```

Name:
Reg.no:

```
    printf("Enter the requests: ");
    for (int i = 0; i < n; i++) {
        scanf("%d", &requests[i]);
    }

    printf("Enter the initial position of the disk head: ");
    scanf("%d", &head);

    FCFS(requests, n, head);

    return 0;
}
```

OUTPUT:



Name:
Reg.no:

(b) SSTF

AIM:

To implement the Shortest Seek Time First (SSTF) disk scheduling algorithm in C language, which selects the disk access request closest to the current head position.

ALGORITHM:

1. **Initialize:**
   - Start with the initial of the disk head.
   - Create an array to store the sequence of requests.
   - Create an array to keep track of whether a request has been serviced.

2. **Calculate Total Movement:**
   - For each request, find the one closest to the current head position that hasn't been serviced yet.
   - Move the head to this closest request and mark it as serviced.
   - Sum the head movements to calculate position the total head movement.

3. **Display the Result:**
   - Print the sequence of requests serviced and the total head movement.

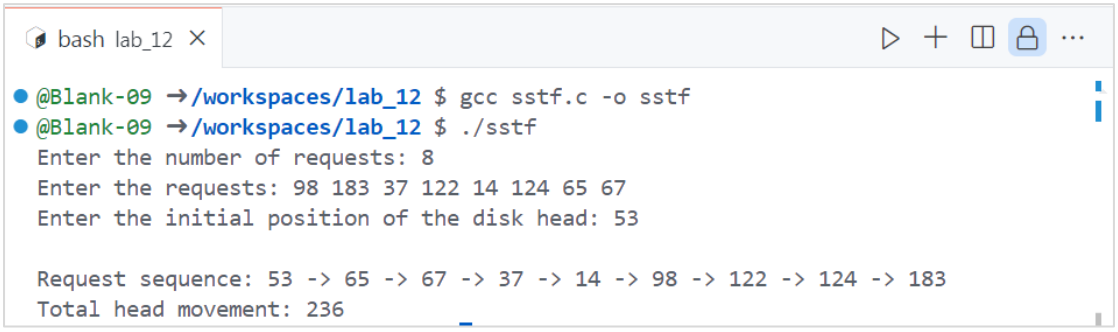
PROGRAM:

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

// Function to calculate absolute difference
int absoluteDifference(int a, int b) {
    return abs(a - b);
}

// Function to find the closest unserviced request
int findClosestRequest(int requests[], int n, int head, int serviced[]) {
    int minDistance = __INT_MAX__;
    int closestRequestIndex = -1;

    for (int i = 0; i < n; i++) {
        if (!serviced[i]) {
            int distance = absoluteDifference(head, requests[i]);
            if (distance < minDistance) {
                minDistance = distance;
                closestRequestIndex = i;
            }
        }
    }

    return closestRequestIndex;
}

void SSTF(int requests[], int n, int head) {
    int totalMovement = 0;
    int currentPosition = head;
    int serviced[n];
```

Name:
Reg.no:

```
    for (int i = 0; i < n; i++) {
        serviced[i] = 0; // Initialize all requests as unserviced
    }

    printf("\nRequest sequence: %d", head);

    for (int i = 0; i < n; i++) {
        int closestRequestIndex = findClosestRequest(requests, n, currentPosition,
serviced);
        int nextRequest = requests[closestRequestIndex];
        totalMovement += absoluteDifference(currentPosition, nextRequest);
        currentPosition = nextRequest;
        serviced[closestRequestIndex] = 1; // Mark this request as serviced
        printf(" -> %d", nextRequest);
    }

    printf("\nTotal head movement: %d\n", totalMovement);
}

int main() {
    int n, head;

    printf("Enter the number of requests: ");
    scanf("%d", &n);

    int requests[n];
    printf("Enter the requests: ");
    for (int i = 0; i < n; i++) {
        scanf("%d", &requests[i]);
    }

    printf("Enter the initial position of the disk head: ");
    scanf("%d", &head);

    SSTF(requests, n, head);

    return 0;
}
```

OUTPUT:



RESULT: The program is successfully compiled and executed. The i/o is verified with the sample i/o.