Bahria University

Karachi Campus

COURSE: CSC-320 OPERATING SYSTEM
TERM: Spring 2020, CLASS: BSE- 4B

SUBMITTED BY:

Muhammad Arbab Anjum
(Name)

02-131182-008
(Enrollment No.)

57226
(Registration No.)

SUBMITTED TO:

Engr. Osama Rehman/ Engr. Fareeha Dilawar

SIGNED ____________ **REMARKS:** ____________ **SCORE:** ____________

INDEX

BAHRIA UNIVERSITY

Karachi Campus

LAB EXPERIMENT NO. 1

LIST OF TASKS

TASK NO	OBJECTIVE
Task 1	Answer the following questions. • In Linux operating system, describe the kernel. • In Linux desktop environment, describe the benefits of virtual desktops • While GUI based tools do exist in Linux, what is the purpose of using the command line interface, i.e. shell? • Use one of the options with the ls command, and describe its usage.
Task 2	By using the command line shell interface, practice the commands given in this lab. Write briefly about the usage of each command.
Task 3	By using gedit, open a text editor and write the C program given below. Save the written file as “hello.c”. In order to compile and execute the output file, do the following: Write down the output of the program below (provide snapshot).
Task 4	Make changes within the above program to display a new output text as given below. Write down the developed program.

Submitted On:

4/2/2020

(Date: DD/MM/YY)

Task 1: Answer the following questions.

Q1. In Linux operating system, describe the kernel.

Answer:-

The Linux kernel is the main component of a Linux operating system (OS) and is the core interface between a computer's hardware and its processes. It communicates between the two, managing resources as efficiently as possible.

The kernel has four jobs:

1. **Memory management:** Keep track of how much memory is used to store what, and where
2. **Process management:** Determine which processes can use the CPU, when, and for how long.
3. **Device drivers:** Act as mediator/interpreter between the hardware and processes
4. **System calls and security:** Receive requests for service from the processes.

Q2. In Linux desktop environment, describe the benefits of virtual desktops.

Answer:-

Desktop virtualization is an increasingly important technology for many organizations. A virtual desktop or virtual desktop infrastructure (VDI) means that a user's desktop environment is stored remotely on a server, rather than on a local PC or other client computing device. Desktop virtualization software separates the desktop operating systems, applications and data from the hardware client, storing this "virtual desktop" on a remote server. The remote server that runs and supports virtual desktops uses software called a hypervisor to create a "virtual machine" that simulates the user's desktop environment and capabilities. In a virtual desktop environment, users access their personal desktop remotely, over the Internet, from any client device.

Desktop virtualization is a valuable technology platform and solves several business problems. With the many benefits desktop virtualization has to offer for an environment, there are four prominent categories that stand out:

1. **Cost Savings**
2. **Simplified Management**
3. **Enhanced Security**
4. **Increased Productivity**

Q3. While GUI based tools do exist in Linux, what is the purpose of using the command line interface, i.e. shell?

Answer:-

A Command Line Interface is a powerful way of user interacting with an operating system. One of the first CLIs was the MS-DOS. It was the operating system for the original personal digital computer (PC), which had been built in the 1980s.

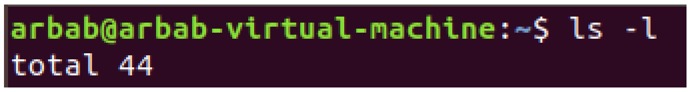
Due to the following reasons command line interface are used:

- If the user knows the correct commands then this type of interface can be much faster than any other type of interface.
- This type of interface needs much less memory (Random Access Memory) in order to use compared to other types of user interfaces.
- This type of interface does not use as much CPU processing time as others
- A low resolution, cheaper monitor can be used with this type of interface.
- A CLI does not require Windows to run.

Q4. Use one of the options with the ls command, and describe its usage.

Answer:-

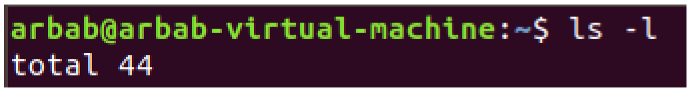
ls command is used to view the contents of a directory. By default, this command will display the contents of your current directory. To display total information about Files/Directories (ls -l):



```
arbab@arbab-virtual-machine:~$ ls -l
total 44
```

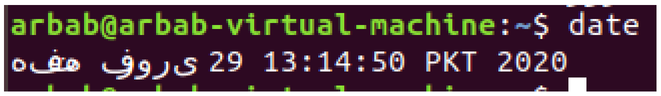
__Task 2__**: By using the command line shell interface, practice the commands given in this lab. Write briefly about the usage of each command.**

ls command:
It is used to list all the contents in the current working directory.
Syntax: $ ls – options <arguments>
Example:-



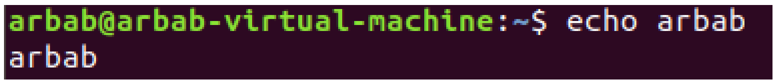
```
arbab@arbab-virtual-machine:~$ ls -l
total 44
```

date Command:
This command is used to display the current date and time.
Syntax:
$date
$date +%ch
Example:-



```
arbab@arbab-virtual-machine:~$ date
ہفتہ فروری 29 13:14:50 PKT 2020
```

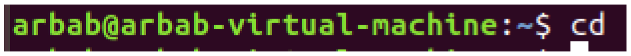
echo Command:
Writes all its parameters to standard output, separated by spaces.
Syntax:
$echo –<options>
Example:-



```
arbab@arbab-virtual-machine:~$ echo arbab
arbab
```

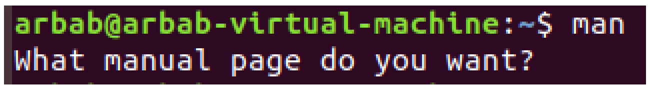
cd Command:

Changes the directory.
Syntax:
$cd [parameter]
Example:-



```
arbab@arbab-virtual-machine:~$ cd
```

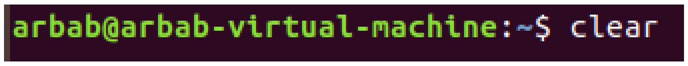
man Command:
This command is used to display text-only manual pages.
Syntax:
$man –<options>
Example:-



```
arbab@arbab-virtual-machine:~$ man
What manual page do you want?
```

clear Command:
This command is used to clear the terminal screen.
Syntax:
$clear

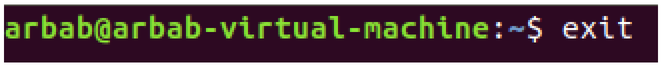
Example:-



```
arbab@arbab-virtual-machine:~$ clear
```

exit Command:
Shell sessions can generally be terminated using this command.
Syntax:
$exit

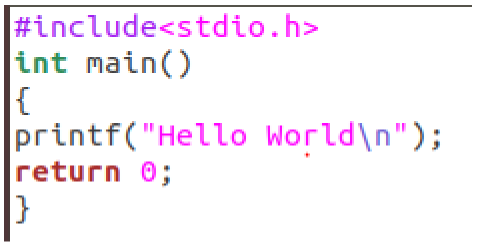
Example:-



```
arbab@arbab-virtual-machine:~$ exit
```

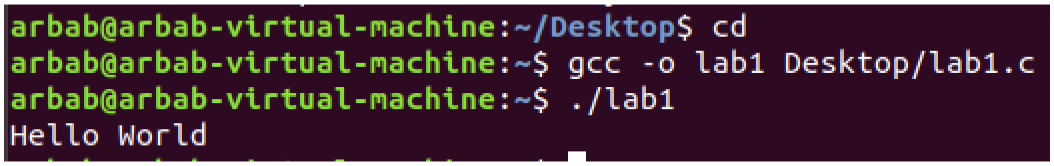
<u>Task 3</u>: By using gedit, open a text editor and write the C program given below. Save the written file as "hello.c". In order to compile and execute the output file, do the following: Write down the output of the program below (provide snapshot).

Code:-



```
#include<stdio.h>
int main()
{
printf("Hello World\n");
return 0;
}
```

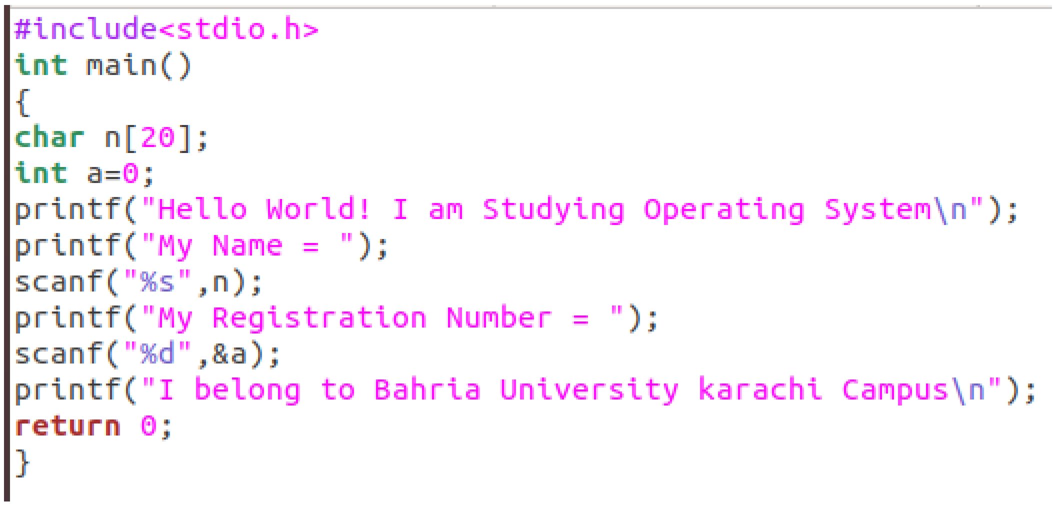
Output:-



```
arbab@arbab-virtual-machine:~/Desktop$ cd
arbab@arbab-virtual-machine:~$ gcc -o lab1 Desktop/lab1.c
arbab@arbab-virtual-machine:~$ ./lab1
Hello World
```

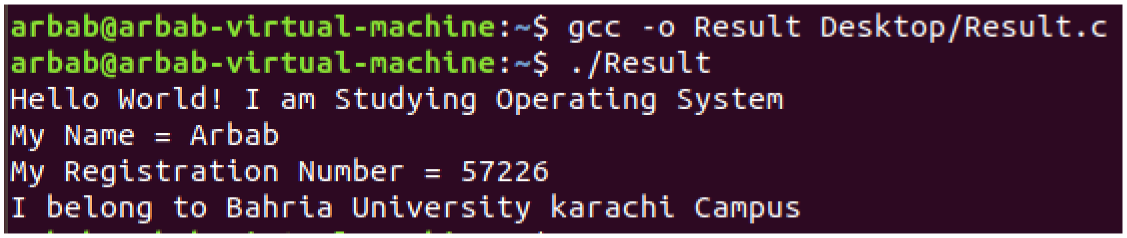
Task 4: Make changes within the above program to display a new output text as given below. Write down the developed program.

Code:-



```c
#include<stdio.h>
int main()
{
char n[20];
int a=0;
printf("Hello World! I am Studying Operating System\n");
printf("My Name = ");
scanf("%s",n);
printf("My Registration Number = ");
scanf("%d",&a);
printf("I belong to Bahria University karachi Campus\n");
return 0;
}
```

Output:-



```
arbab@arbab-virtual-machine:~$ gcc -o Result Desktop/Result.c
arbab@arbab-virtual-machine:~$ ./Result
Hello World! I am Studying Operating System
My Name = Arbab
My Registration Number = 57226
I belong to Bahria University karachi Campus
```

BAHRIA UNIVERSITY

Karachi Campus

LAB EXPERIMENT NO. 2

LIST OF TASKS

TASK NO	OBJECTIVE
Task 1	Practice all the Linux commands discussed in this lab while taking assistance using the man command. Write the complete syntax used for utilizing the cp, mv and rm commands in Linux shell.
Task 2	Write a shell script to display your address over multiple lines.
Task 3	Write a shell script that would traverse among any three directories that are placed under the /home directory. While moving from one directory to another, the script should display the name of the current working directory and list the content within that directory, including the hidden files.
Task 4	Write the C programs provided in this lab and generate their outputs over Linux environment (provide snapshot).

Submitted On:

12/2/2020

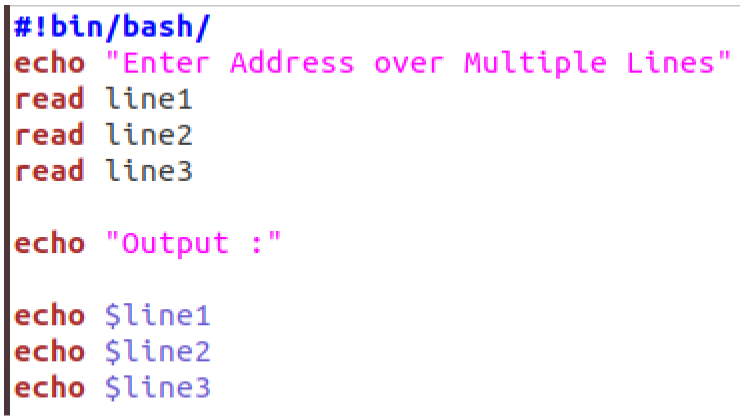
(Date: DD/MM/YY)

Task 1: Practice all the Linux commands discussed in this lab while taking assistance using the man command. Write the complete syntax used for utilizing the cp, mv and rm commands in Linux shell.

cp	Copy file or directory
mv	Move file or directory
rm	Remove file or directory

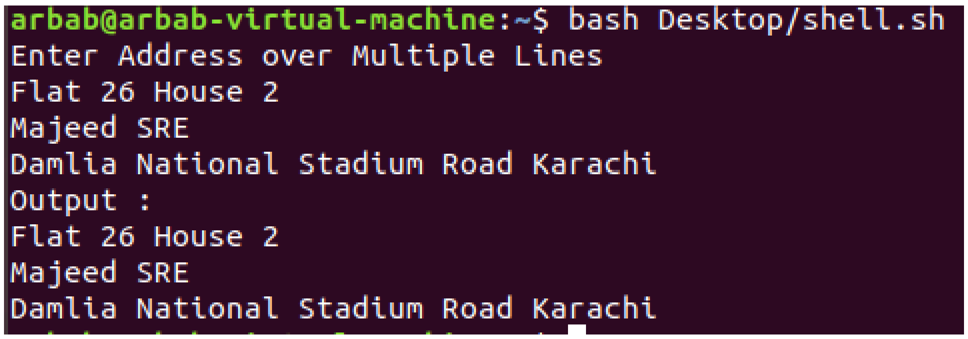
Task 2: Write a shell script to display your address over multiple lines.

Code:-



```
#!bin/bash/
echo "Enter Address over Multiple Lines"
read line1
read line2
read line3

echo "Output :"

echo $line1
echo $line2
echo $line3
```

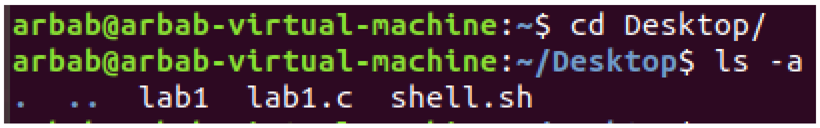
Output:-



```
arbab@arbab-virtual-machine:~$ bash Desktop/shell.sh
Enter Address over Multiple Lines
Flat 26 House 2
Majeed SRE
Damlia National Stadium Road Karachi
Output :
Flat 26 House 2
Majeed SRE
Damlia National Stadium Road Karachi
```

Task 3: Write a shell script that would traverse among any three directories that are placed under the /home directory. While moving from one directory to another, the script should display the name of the current working directory and list the content within that directory, including the hidden files.

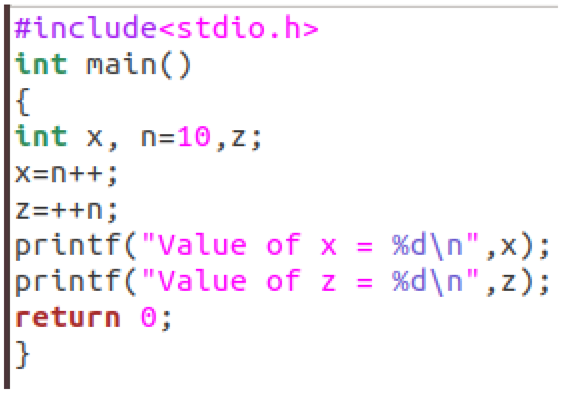
Output:-



```
arbab@arbab-virtual-machine:~$ cd Desktop/
arbab@arbab-virtual-machine:~/Desktop$ ls -a
.  ..  lab1  lab1.c  shell.sh
```

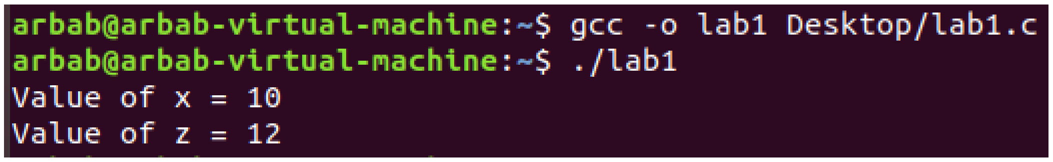
Task 4: Write the C programs provided in this lab and generate their outputs over Linux environment (provide snapshot).

Code:-



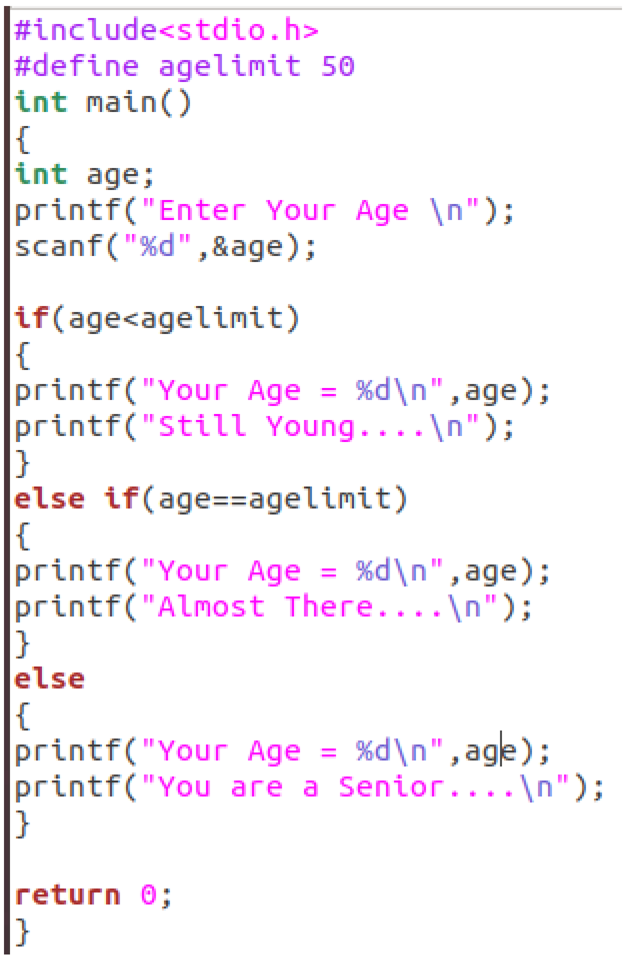
```
#include<stdio.h>
int main()
{
int x, n=10,z;
x=n++;
z=++n;
printf("Value of x = %d\n",x);
printf("Value of z = %d\n",z);
return 0;
}
```

Output:-



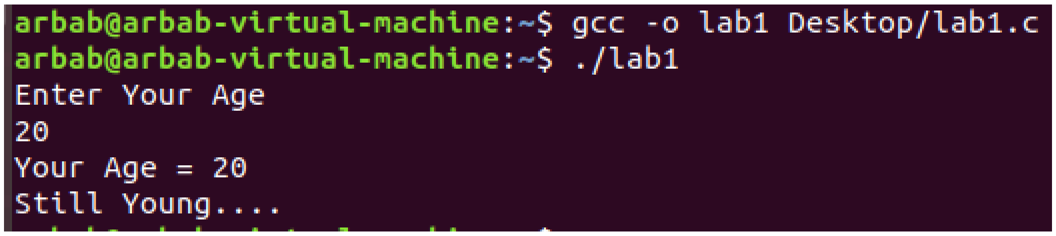
```
arbab@arbab-virtual-machine:~$ gcc -o lab1 Desktop/lab1.c
arbab@arbab-virtual-machine:~$ ./lab1
Value of x = 10
Value of z = 12
```

Code:-



```c
#include<stdio.h>
#define agelimit 50
int main()
{
int age;
printf("Enter Your Age \n");
scanf("%d",&age);

if(age<agelimit)
{
printf("Your Age = %d\n",age);
printf("Still Young....\n");
}
else if(age==agelimit)
{
printf("Your Age = %d\n",age);
printf("Almost There....\n");
}
else
{
printf("Your Age = %d\n",age);
printf("You are a Senior....\n");
}

return 0;
}
```

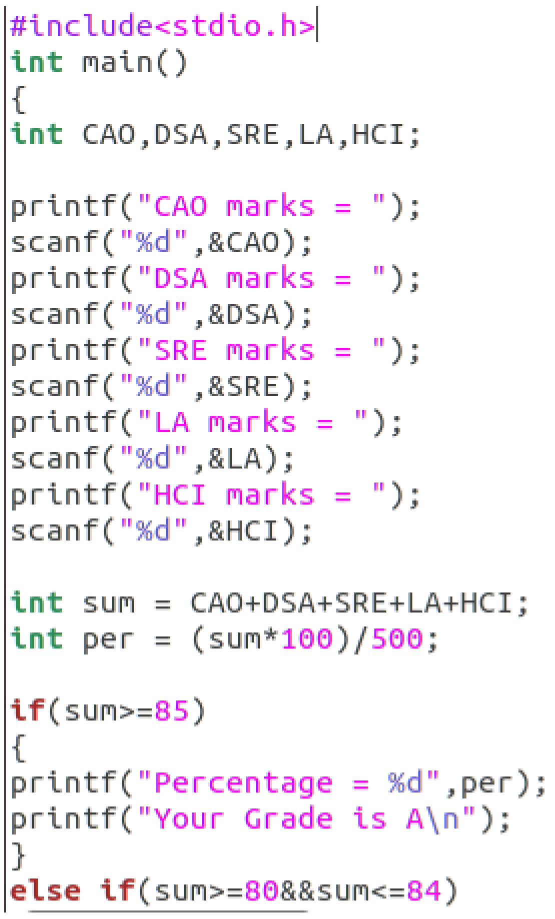
Output:-



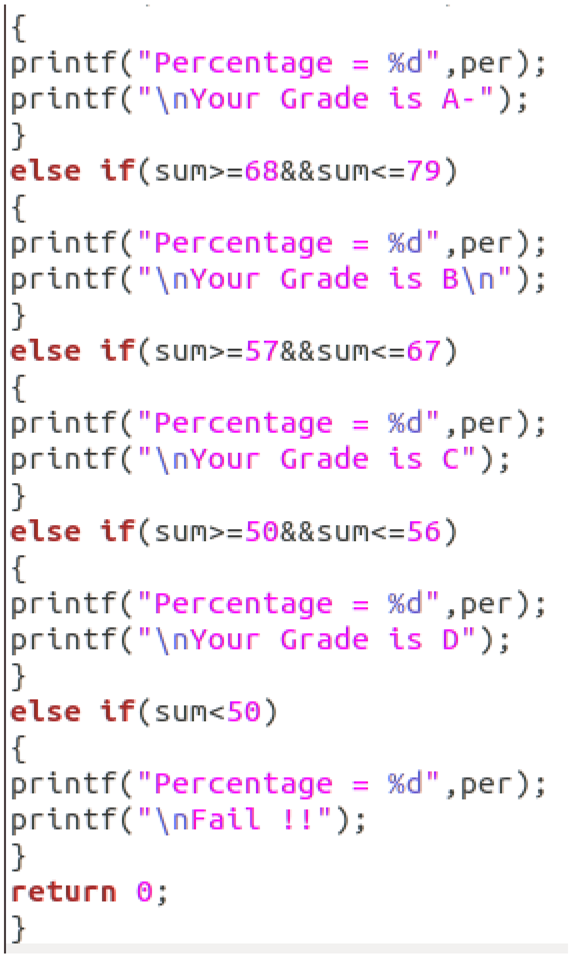
```
arbab@arbab-virtual-machine:~$ gcc -o lab1 Desktop/lab1.c
arbab@arbab-virtual-machine:~$ ./lab1
Enter Your Age
20
Your Age = 20
Still Young....
```

Task 5: **Write a C program on the Linux environment that takes your marks as an input and display your grades accordingly to that followed at Bahria University. Limit your program to a maximum of five subjects. Use the suitable logical operator(s), i.e. and (&&), or (||), not (!), if required.**

Code:-

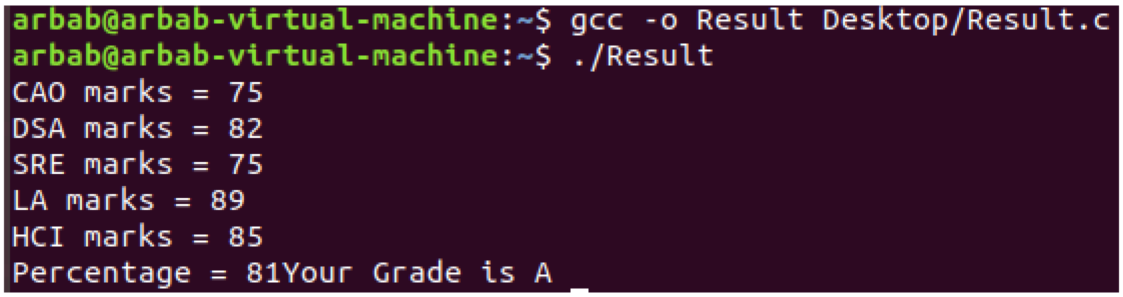


```c
#include<stdio.h>
int main()
{
int CAO,DSA,SRE,LA,HCI;

printf("CAO marks = ");
scanf("%d",&CAO);
printf("DSA marks = ");
scanf("%d",&DSA);
printf("SRE marks = ");
scanf("%d",&SRE);
printf("LA marks = ");
scanf("%d",&LA);
printf("HCI marks = ");
scanf("%d",&HCI);

int sum = CAO+DSA+SRE+LA+HCI;
int per = (sum*100)/500;

if(sum>=85)
{
printf("Percentage = %d",per);
printf("Your Grade is A\n");
}
else if(sum>=80&&sum<=84)
```



```c
{
printf("Percentage = %d",per);
printf("\nYour Grade is A-");
}
else if(sum>=68&&sum<=79)
{
printf("Percentage = %d",per);
printf("\nYour Grade is B\n");
}
else if(sum>=57&&sum<=67)
{
printf("Percentage = %d",per);
printf("\nYour Grade is C");
}
else if(sum>=50&&sum<=56)
{
printf("Percentage = %d",per);
printf("\nYour Grade is D");
}
else if(sum<50)
{
printf("Percentage = %d",per);
printf("\nFail !!");
}
return 0;
}
```

Output:-



```
arbab@arbab-virtual-machine:~$ gcc -o Result Desktop/Result.c
arbab@arbab-virtual-machine:~$ ./Result
CAO marks = 75
DSA marks = 82
SRE marks = 75
LA marks = 89
HCI marks = 85
Percentage = 81Your Grade is A
```

BAHRIA UNIVERSITY

Karachi Campus

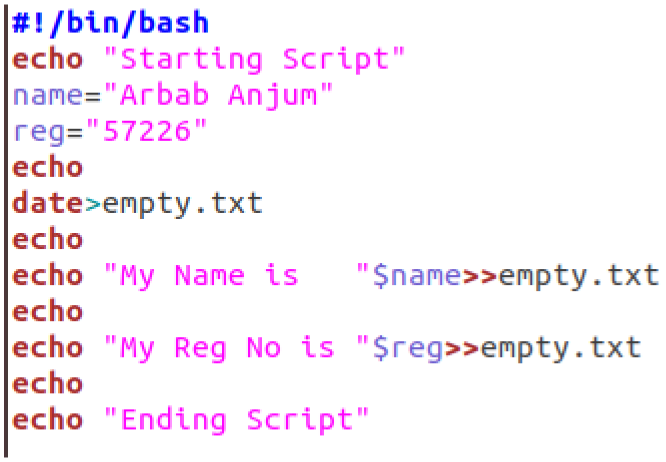
LAB EXPERIMENT NO. 3
LIST OF TASKS

TASK NO	OBJECTIVE
Task 1	Create an empty file with a .txt extension. Write a shell script that would write the current date, student's name and registration number into that file, while using variables for all three entries.
Task 2	Create a .txt file and input ten lines of entry while mixing it with both alphanumeric characters. Sort the contents of the created file in an ascending order and write the sorted output into another file.
Task 3	Write two different C language programs that would generate the following for a given number x, e.g.$x = 30$: 1. Calculate the factorial of x. 2. Calculate the Fibonacci series (0, 1, 1, 2, 3, 5, 8...) up to x. Compile and run both programs using a single shell script, while having a running gap of 5 seconds between the first and second program. The generated output should properly display on which program is currently running (tip use *echo* command).

Submitted On:
19/2/2020
(Date: DD/MM/YY)

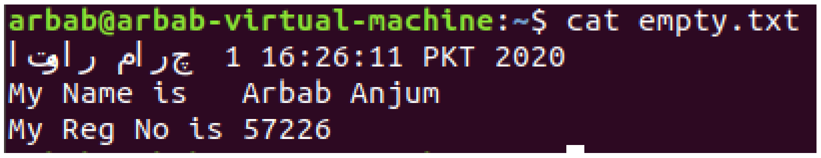
Task 1: Create an empty file with a .txt extension. Write a shell script that would write the current date, student's name and registration number into that file, while using variables for all three entries.

Code:-



```
#!/bin/bash
echo "Starting Script"
name="Arbab Anjum"
reg="57226"
echo
date>empty.txt
echo
echo "My Name is   "$name>>empty.txt
echo
echo "My Reg No is "$reg>>empty.txt
echo
echo "Ending Script"
```

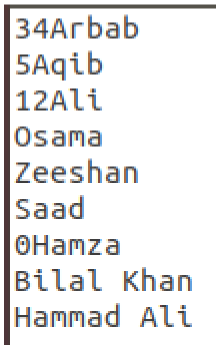
Output:-



```
arbab@arbab-virtual-machine:~$ cat empty.txt
اتوار مارچ  1 16:26:11 PKT 2020
My Name is   Arbab Anjum
My Reg No is 57226
```

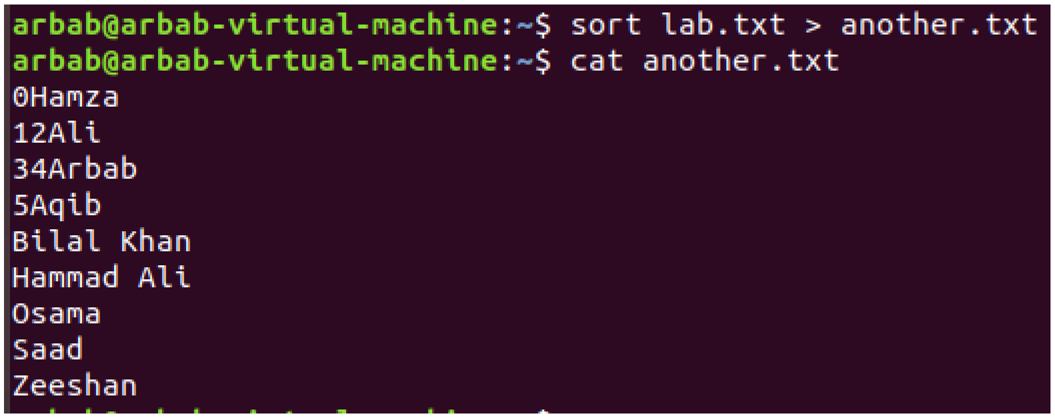
Task 2: Create a .txt file and input ten lines of entry while mixing it with both alphanumeric characters. Sort the contents of the created file in an ascending order and write the sorted output into another file.

Code:-



```
34Arbab
5Aqib
12Ali
Osama
Zeeshan
Saad
0Hamza
Bilal Khan
Hammad Ali
```

Output:-



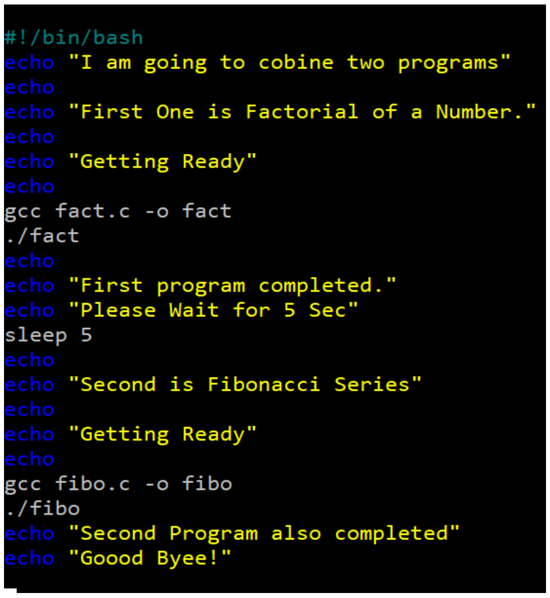
```
arbab@arbab-virtual-machine:~$ sort lab.txt > another.txt
arbab@arbab-virtual-machine:~$ cat another.txt
0Hamza
12Ali
34Arbab
5Aqib
Bilal Khan
Hammad Ali
Osama
Saad
Zeeshan
```

Task 3: Write two different C language programs that would generate the following for a given number x, e.g. $x = 30$:

3. **Calculate the factorial of x.**
4. **Calculate the Fibonacci series (0, 1, 1, 2, 3, 5, 8...) up to x.**

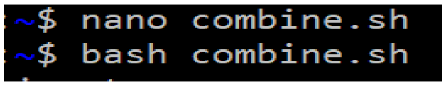
Compile and run both programs using a single shell script, while having a running gap of 5 seconds between the first and second program. The generated output should properly display on which program is currently running (tip use *echo* command).

Code:

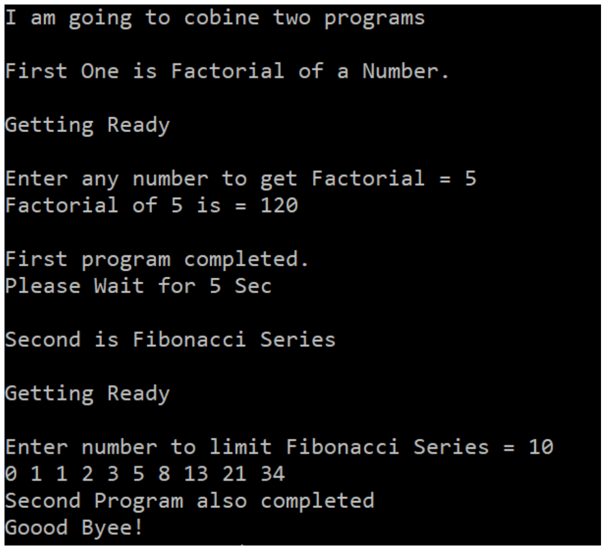


```
#!/bin/bash
echo "I am going to cobine two programs"
echo
echo "First One is Factorial of a Number."
echo
echo "Getting Ready"
echo
gcc fact.c -o fact
./fact
echo
echo "First program completed."
echo "Please Wait for 5 Sec"
sleep 5
echo
echo "Second is Fibonacci Series"
echo
echo "Getting Ready"
echo
gcc fibo.c -o fibo
./fibo
echo "Second Program also completed"
echo "Goood Byee!"
```

Output:



```
~$ nano combine.sh
~$ bash combine.sh
```



```
I am going to cobine two programs

First One is Factorial of a Number.

Getting Ready

Enter any number to get Factorial = 5
Factorial of 5 is = 120

First program completed.
Please Wait for 5 Sec

Second is Fibonacci Series

Getting Ready

Enter number to limit Fibonacci Series = 10
0 1 1 2 3 5 8 13 21 34
Second Program also completed
Goood Byee!
```

BAHRIA UNIVERSITY

Karachi Campus

LAB EXPERIMENT NO. 4

LIST OF TASKS

TASK NO	OBJECTIVE
Task 1	Write a single shell script that creates four different files, while taking the names of all created files as input from the user. As the files contents, insert your name in the first file, registration number in the second and section details in the third. These should be followed by merging the contents of all three files into the fourth one.
Task 2	Write a shell script that creates a “Files Location Log”. The paths of all files, having the same extension, should be stored in one log. The file extension should be taken as an input from the user, and the created logs should be named as “mylog_extension.txt”, where “extension” is that taken as input from the user. The search process should be for all file in the system, starting from the root directory (/). All log files of different file extension should be stored inside a single directory by the name of “mylog” that would be present at your home directory.
Task 3	Write a shell script that either performs a file sort, file search or directory listing operation based on the user’s selection of the operation he/she would like to execute.
Task 4	Write a C program that takes values of two matrices of size $(m\times1)$ and $(1\times n)$ as input from the user. Multiply the above two matrixes and store the resulting $(m\times n)$ matrix in a 2D array. Display the contents of the first and second matrices and also the resulting matrix. Achieve alignment in the displayed content as much possible.

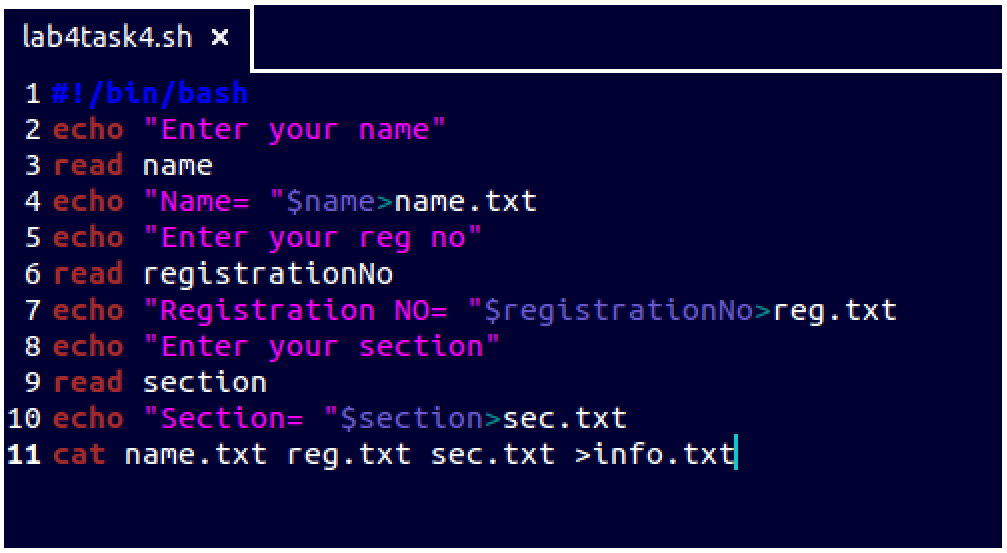
Submitted On:

26/2/2020

(Date: DD/MM/YY)

Task1: Write a single shell script that creates four different files, while taking the names of all created files as input from the user. As the files contents, insert your name in the first file, registration number in the second and section details in the third. These should be followed by merging the contents of all three files into the fourth one.

Code:

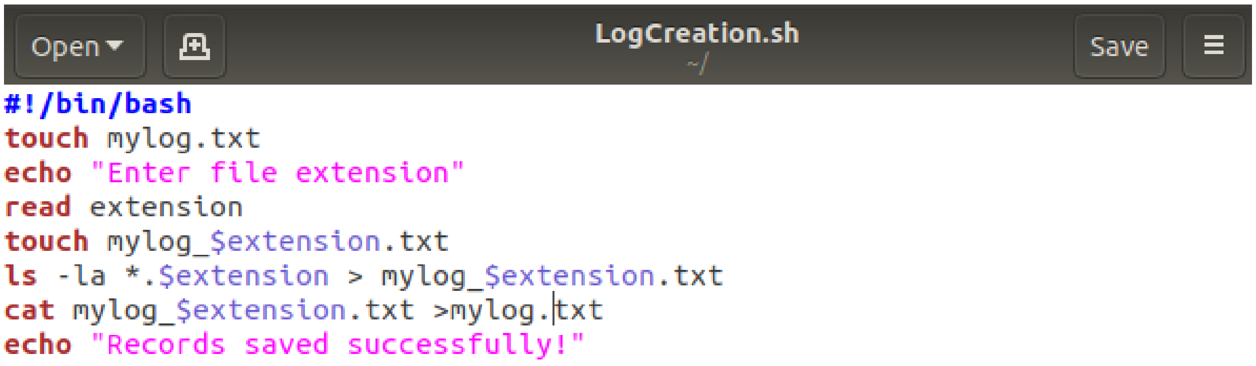


lab4task4.sh ×

```
#!/bin/bash
echo "Enter your name"
read name
echo "Name= "$name>name.txt
echo "Enter your reg no"
read registrationNo
echo "Registration NO= "$registrationNo>reg.txt
echo "Enter your section"
read section
echo "Section= "$section>sec.txt
cat name.txt reg.txt sec.txt >info.txt
```

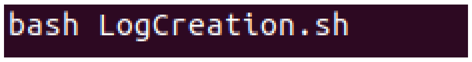
Task2: Write a shell script that creates a “Files Location Log”. The paths of all files, having the same extension, should be stored in one log. The file extension should be taken as an input from the user, and the created logs should be named as “mylog_extension.txt”, where “extension” is that taken as input from the user. The search process should be for all file in the system, starting from the root directory (/). All log files of different file extension should be stored inside a single directory by the name of “mylog” that would be present at your home directory.

Code:

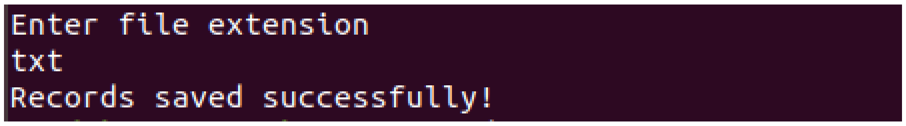


```
#!/bin/bash
touch mylog.txt
echo "Enter file extension"
read extension
touch mylog_$extension.txt
ls -la *.$extension > mylog_$extension.txt
cat mylog_$extension.txt >mylog.txt
echo "Records saved successfully!"
```

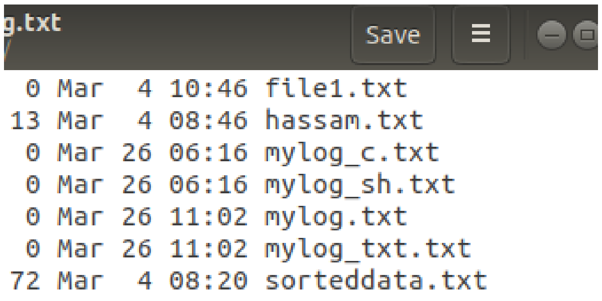
Output:



```
bash LogCreation.sh
```



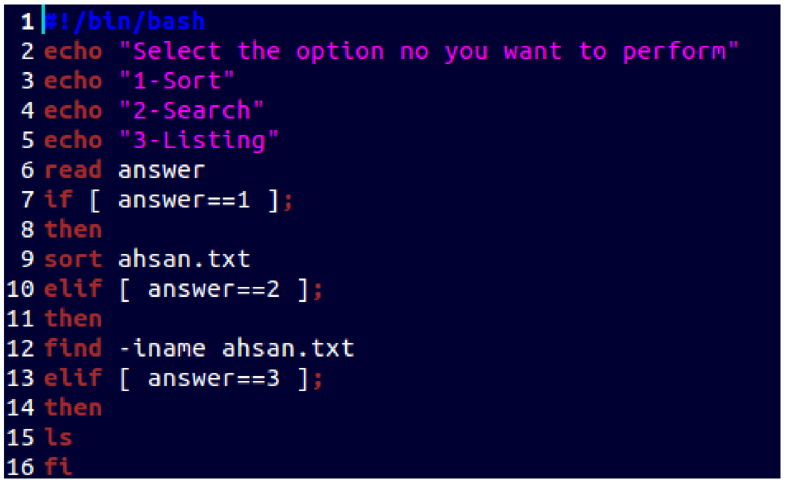
```
Enter file extension
txt
Records saved successfully!
```



```
 0 Mar  4 10:46 file1.txt
13 Mar  4 08:46 hassam.txt
 0 Mar 26 06:16 mylog_c.txt
 0 Mar 26 06:16 mylog_sh.txt
 0 Mar 26 11:02 mylog.txt
 0 Mar 26 11:02 mylog_txt.txt
72 Mar  4 08:20 sorteddata.txt
```

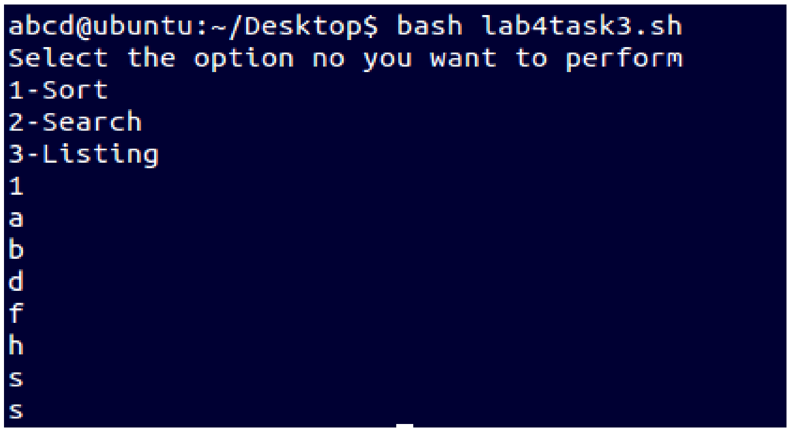
Task3: Write a shell script that either performs a file sort, file search or directory listing operation based on the user's selection of the operation he/she would like to execute.

Code:



```
#!/bin/bash
echo "Select the option no you want to perform"
echo "1-Sort"
echo "2-Search"
echo "3-Listing"
read answer
if [ answer==1 ];
then
sort ahsan.txt
elif [ answer==2 ];
then
find -iname ahsan.txt
elif [ answer==3 ];
then
ls
fi
```

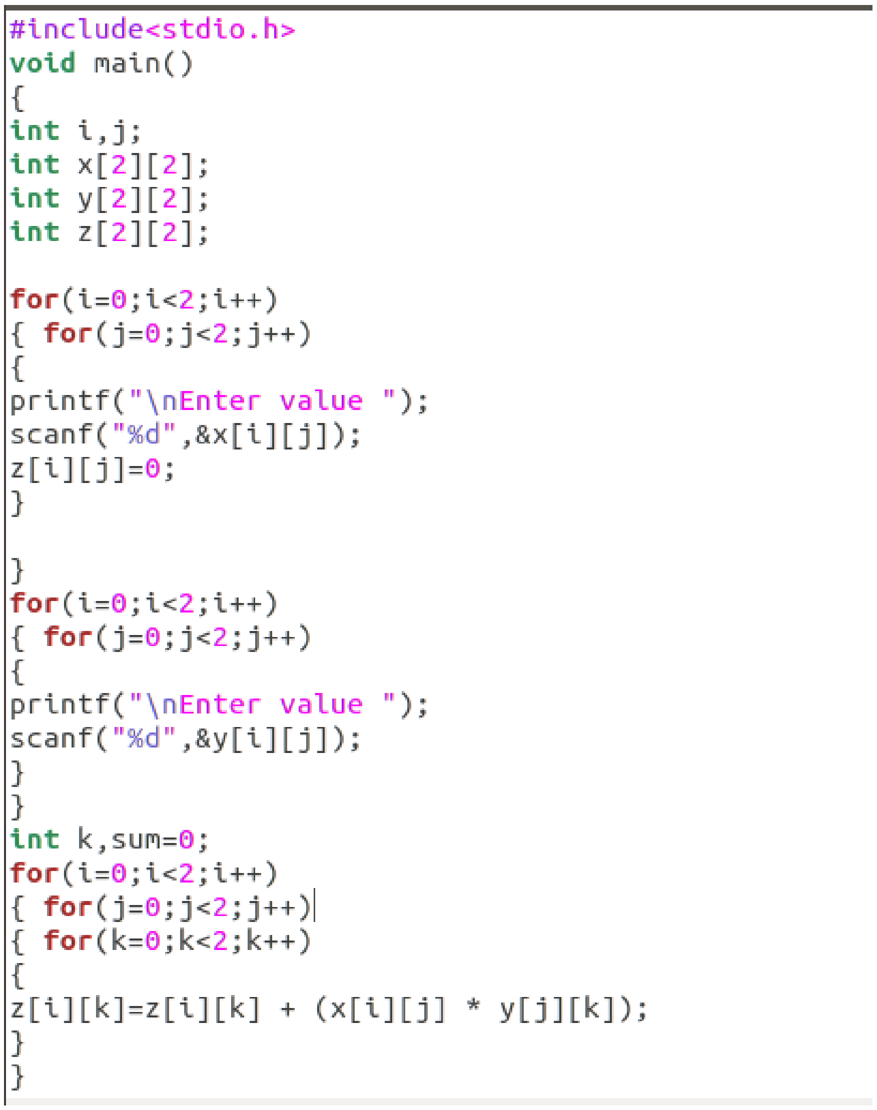
Output:



```
abcd@ubuntu:~/Desktop$ bash lab4task3.sh
Select the option no you want to perform
1-Sort
2-Search
3-Listing
1
a
b
d
f
h
s
s
```

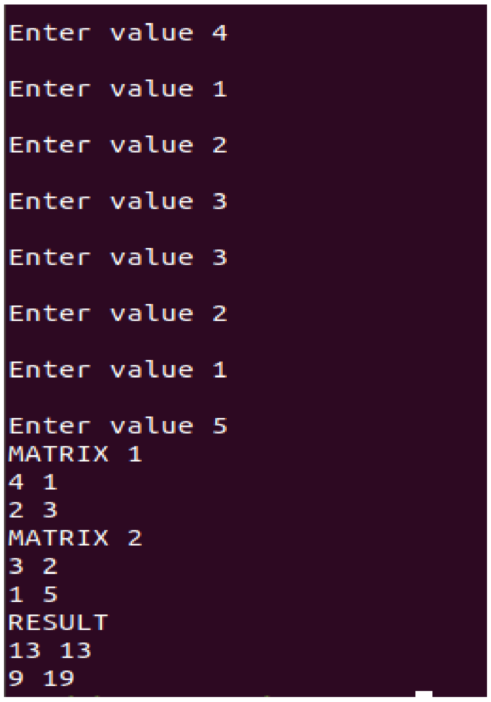
Task4: Write a C program that takes values of two matrices of size (m×1) and (1×n) as input from the user. Multiply the above two matrixes and store the resulting (m×n) matrix in a 2D array. Display the contents of the first and second matrices and also the resulting matrix. Achieve alignment in the displayed content as much possible.

Code:



```
#include<stdio.h>
void main()
{
int i,j;
int x[2][2];
int y[2][2];
int z[2][2];

for(i=0;i<2;i++)
{ for(j=0;j<2;j++)
{
printf("\nEnter value ");
scanf("%d",&x[i][j]);
z[i][j]=0;
}

}
for(i=0;i<2;i++)
{ for(j=0;j<2;j++)
{
printf("\nEnter value ");
scanf("%d",&y[i][j]);
}
}
int k,sum=0;
for(i=0;i<2;i++)
{ for(j=0;j<2;j++)
{ for(k=0;k<2;k++)
{
z[i][k]=z[i][k] + (x[i][j] * y[j][k]);
}
}
}
printf("MATRIX 1\n");
for(i=0;i<2;i++)
{ for(j=0;j<2;j++)
{
printf("%d ",x[i][j]);
}
printf("\n");
}

printf("MATRIX 2\n");
for(i=0;i<2;i++)
{ for(j=0;j<2;j++)
{
printf("%d ",y[i][j]);
}
printf("\n");
}

printf("RESULT\n");
for(i=0;i<2;i++)
{ for(j=0;j<2;j++)
{
printf("%d ",z[i][j]);
}
printf("\n");
}
}
```

Output:



```
Enter value 4

Enter value 1

Enter value 2

Enter value 3

Enter value 3

Enter value 2

Enter value 1

Enter value 5
MATRIX 1
4 1
2 3
MATRIX 2
3 2
1 5
RESULT
13 13
9 19
```

BAHRIA UNIVERSITY

Karachi Campus

LAB EXPERIMENT NO. 5

LIST OF TASKS

TASK NO	OBJECTIVE
Task 1	While utilizing the Linux commands studied so far, provide an example (other than the one shown in this Lab) of a combination of several Linux commands in which pipes are used more than once. Also provide a snapshot of the generated output.
Task 2	Write a shell script that records the full path of all the files present within a directory into a record.txt file. Along with full path name, the script should also record the number of words, characters and lines within each file.
Task 3	Write a C program that asks the user to provide an integer input in the main() function. The program would call a function even_odd() from the main() function, where the function even_odd() accepts an integer as an argument, determine and display if the passed integer is either even or odd.

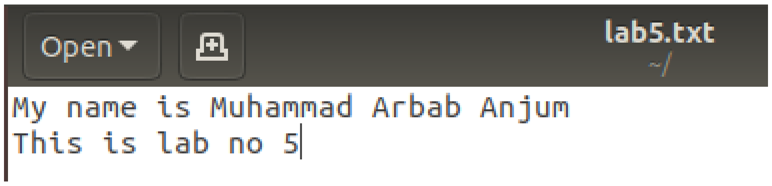
Submitted On:

5/3/2020

(Date: DD/MM/YY)

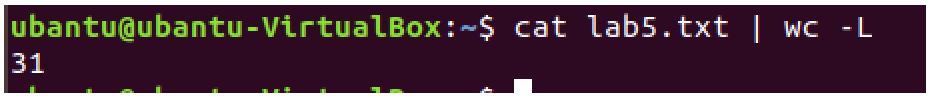
<u>Task 1</u>: While utilizing the Linux commands studied so far, provide an example *(other than the one shown in this Lab)* of a combination of several Linux commands in which pipes are used more than once. Also provide a snapshot of the generated output.

CODE:



```
My name is Muhammad Arbab Anjum
This is lab no 5
```

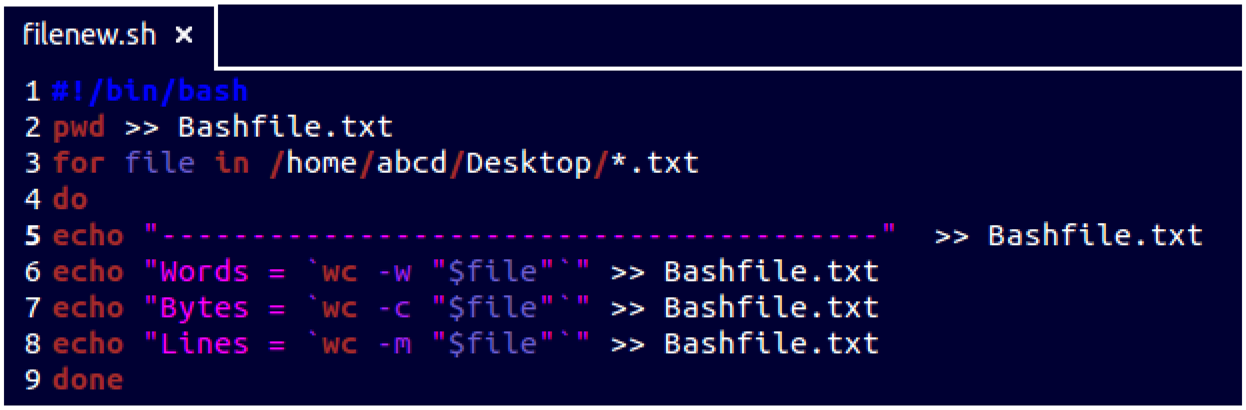
OUTPUT:



```
ubantu@ubantu-VirtualBox:~$ cat lab5.txt | wc -L
31
```

Task 2: Write a shell script that records the full path of all the files present within a directory into a record.txt file. Along with full path name, the script should also record the number of words, characters and lines within each file.

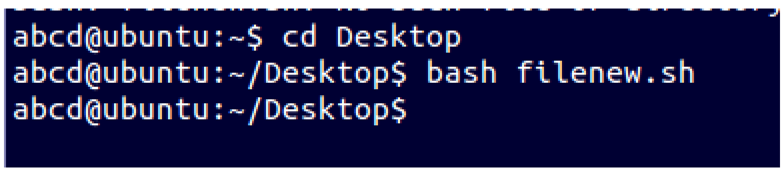
CODE:



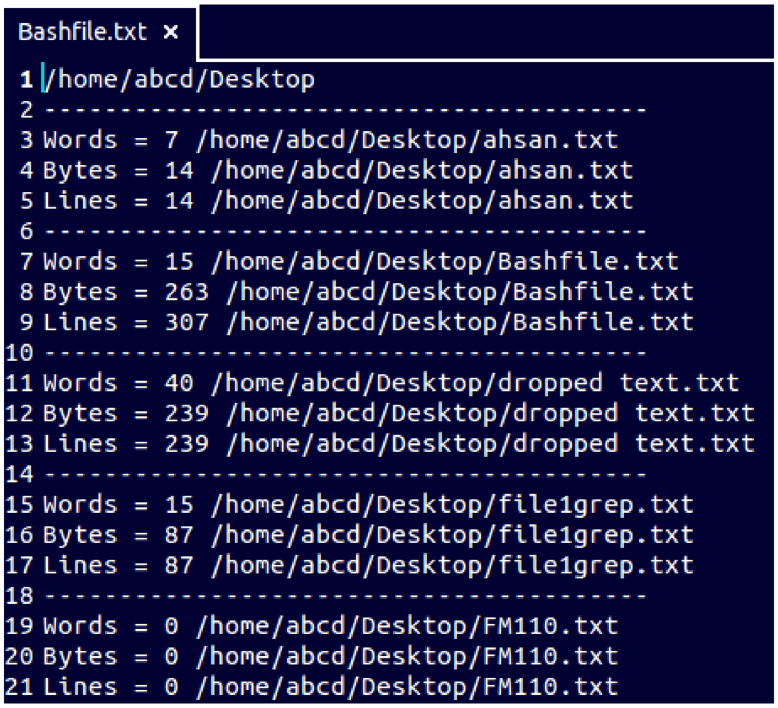
filenew.sh ×

```bash
#!/bin/bash
pwd >> Bashfile.txt
for file in /home/abcd/Desktop/*.txt
do
echo "----------------------------------------"  >> Bashfile.txt
echo "Words = `wc -w "$file"`" >> Bashfile.txt
echo "Bytes = `wc -c "$file"`" >> Bashfile.txt
echo "Lines = `wc -m "$file"`" >> Bashfile.txt
done
```

OUTPUT:



```
abcd@ubuntu:~$ cd Desktop
abcd@ubuntu:~/Desktop$ bash filenew.sh
abcd@ubuntu:~/Desktop$
```

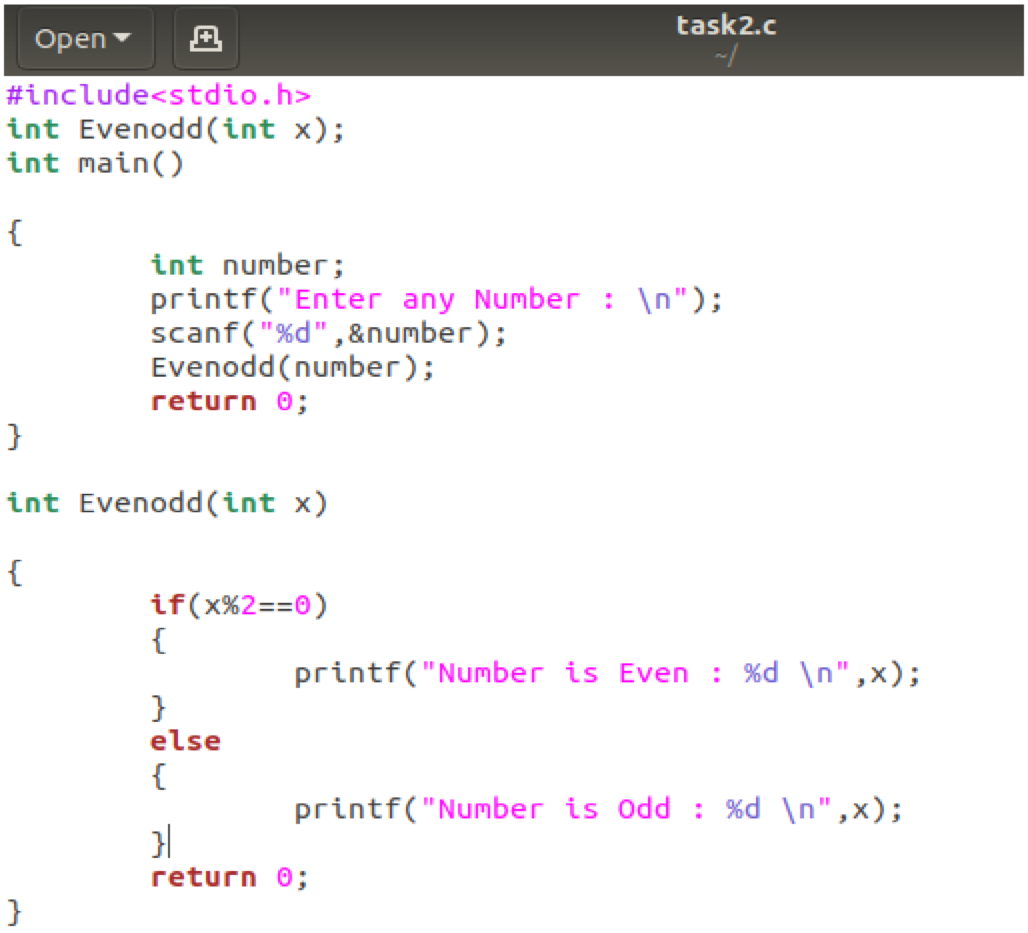


Bashfile.txt ×

```
/home/abcd/Desktop
----------------------------------------
Words = 7 /home/abcd/Desktop/ahsan.txt
Bytes = 14 /home/abcd/Desktop/ahsan.txt
Lines = 14 /home/abcd/Desktop/ahsan.txt
----------------------------------------
Words = 15 /home/abcd/Desktop/Bashfile.txt
Bytes = 263 /home/abcd/Desktop/Bashfile.txt
Lines = 307 /home/abcd/Desktop/Bashfile.txt
----------------------------------------
Words = 40 /home/abcd/Desktop/dropped text.txt
Bytes = 239 /home/abcd/Desktop/dropped text.txt
Lines = 239 /home/abcd/Desktop/dropped text.txt
----------------------------------------
Words = 15 /home/abcd/Desktop/file1grep.txt
Bytes = 87 /home/abcd/Desktop/file1grep.txt
Lines = 87 /home/abcd/Desktop/file1grep.txt
----------------------------------------
Words = 0 /home/abcd/Desktop/FM110.txt
Bytes = 0 /home/abcd/Desktop/FM110.txt
Lines = 0 /home/abcd/Desktop/FM110.txt
```

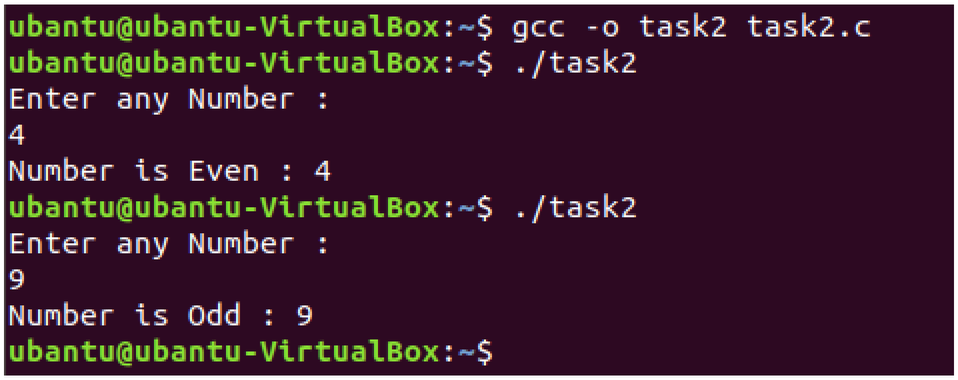
Task 3: Write a C program that asks the user to provide an integer input in the *main()* function. The program would call a function *even_odd()* from the *main()* function, where the function *even_odd()* accepts an integer as an argument, determine and display if the passed integer is either even or odd.

CODE:



```c
#include<stdio.h>
int Evenodd(int x);
int main()

{
	int number;
	printf("Enter any Number : \n");
	scanf("%d",&number);
	Evenodd(number);
	return 0;
}

int Evenodd(int x)

{
	if(x%2==0)
	{
		printf("Number is Even : %d \n",x);
	}
	else
	{
		printf("Number is Odd : %d \n",x);
	}
	return 0;
}
```

OUTPUT:



```
ubantu@ubantu-VirtualBox:~$ gcc -o task2 task2.c
ubantu@ubantu-VirtualBox:~$ ./task2
Enter any Number :
4
Number is Even : 4
ubantu@ubantu-VirtualBox:~$ ./task2
Enter any Number :
9
Number is Odd : 9
ubantu@ubantu-VirtualBox:~$
```

BAHRIA UNIVERSITY

Karachi Campus

LAB EXPERIMENT NO.6

LIST OF TASKS

TASK NO	OBJECTIVE
Task 1	Write what you have learned in few lines on each of the three programs that were using the *fork()* system call.
Task 2	Write a C program that uses *fork()* system call to print a single line eight times without using *for* loop and repeated *printf* command.
Task 3	Code the C program given below and explain what it does along with providing a snapshot of the output. Investigate and write about the usage of *execlp()* system call.

Submitted On:

12/3/2020

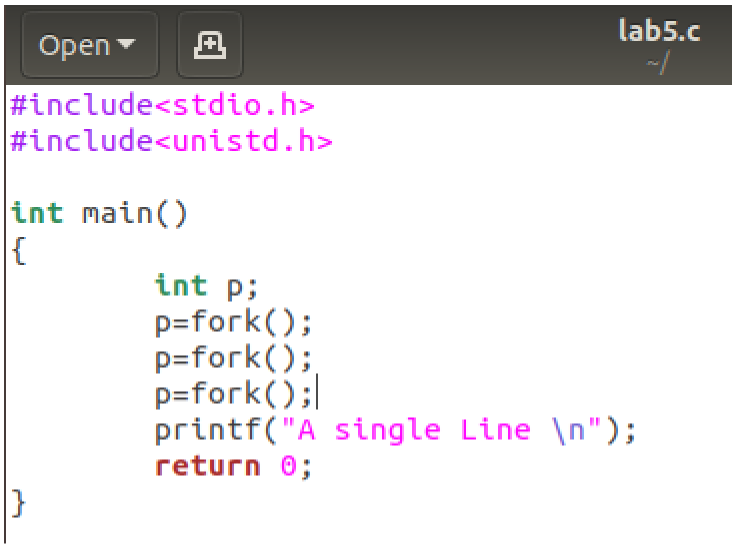
(Date: DD/MM/YY)

__Task 1__: **Write what you have learned in few lines on each of the three programs that were using the *fork()* system call.**

1. Fork() method is used to know about the parent and child process of current process. Fork() returns **0** if it is in a child class otherwise if it is in parent process it will return its child process's **id**. If there is no child of the current process, it will return **-1**.
2. Suppose there are three process A, B and C. Process A is parent of process B and B is the parent of C.
   Now, if we use fork in process A, it will return id of process B. Similarly, in process B, the id of C will be returned. In C, zero will be returned.
3. Another use of fork() is the running of statements, following the fork, multiple times. It works as the power of 2. One fork will run the statement twice. Two forks will run the statements 4 times, as 2 raises to the power 2. Similarly, if we want to run a statement 16 times, we will use fork 4 times.

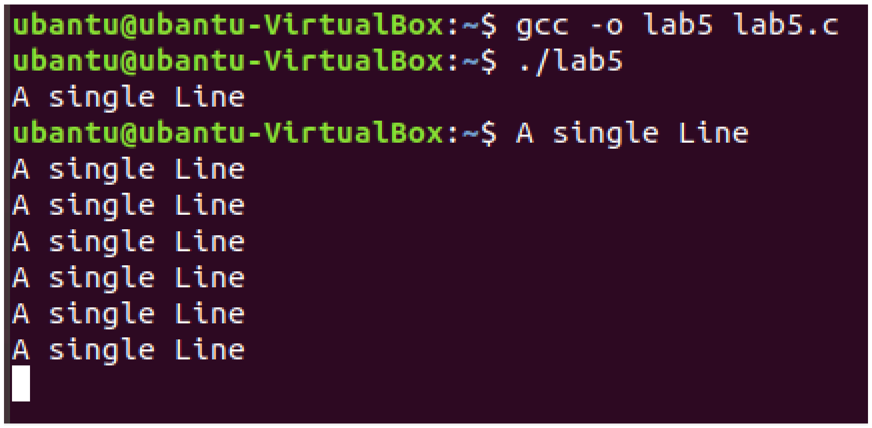
Task 2: **Write a C program that uses *fork()* system call to print a single line eight times without using *for* loop and repeated *printf* command.**

CODE:



```c
#include<stdio.h>
#include<unistd.h>

int main()
{
        int p;
        p=fork();
        p=fork();
        p=fork();
        printf("A single Line \n");
        return 0;
}
```

OUTPUT:



```
ubantu@ubantu-VirtualBox:~$ gcc -o lab5 lab5.c
ubantu@ubantu-VirtualBox:~$ ./lab5
A single Line
ubantu@ubantu-VirtualBox:~$ A single Line
A single Line
A single Line
A single Line
A single Line
A single Line
A single Line
```

Task 3: **Code the C program given below and explain what it does along with providing a snapshot of the output. Investigate and write about the usage of *execlp()* system call.**

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

// Required by for routine
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>

main()
{
        int pid;
        pid = fork();

         if (pid < 0)
        { // error occurred
                fprintf(stderr, "Fork failed!\n");
                exit(-1);
        }
        else if (pid == 0)
        { // child process
                printf("I am the child, return from fork=%d\n", pid);
                execlp("/bin/ls", "ls", NULL);
        }
        else
        { // parent process
                printf("I am the parent, return from fork, child pid=%d\n", pid);
        }
}
```

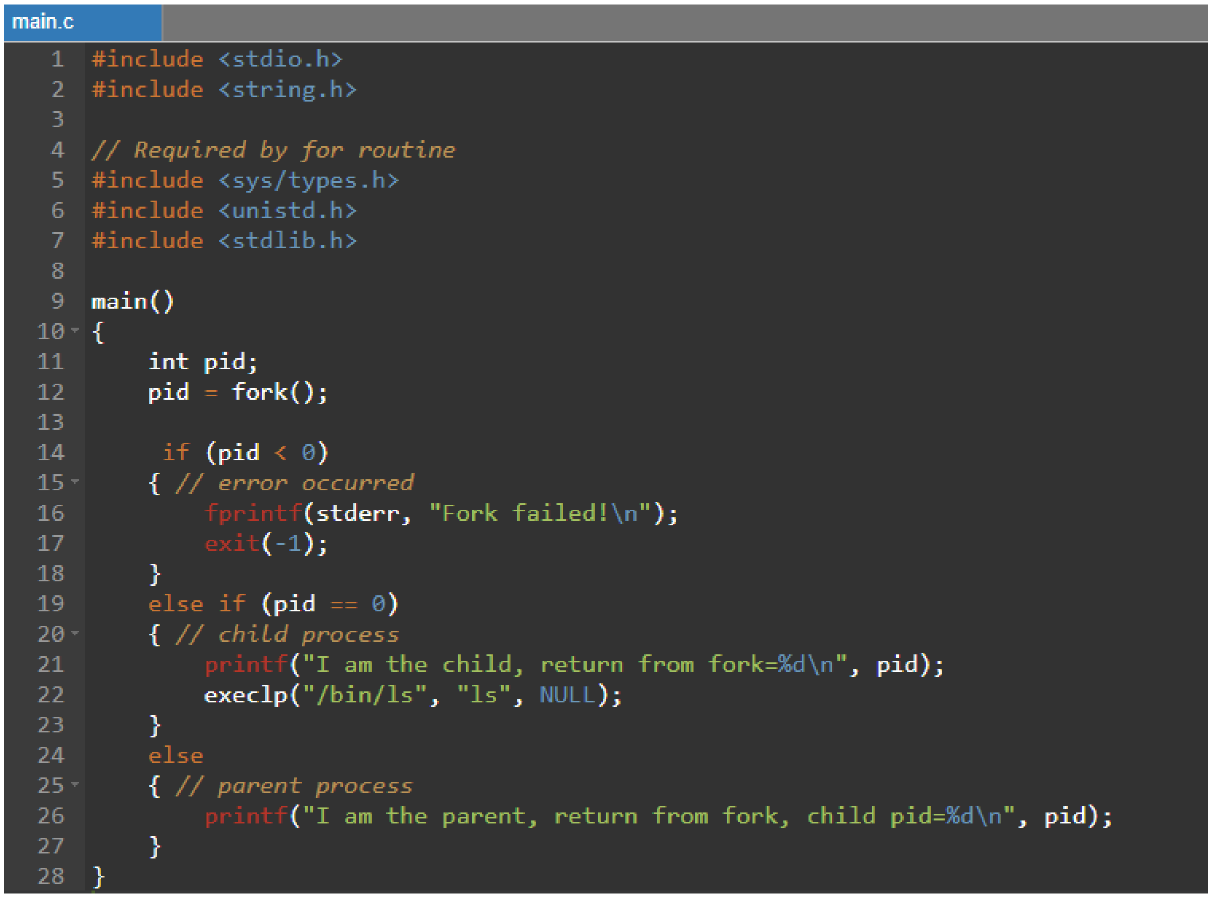
Explanation:

The execlp() family of commands can be used to execute an application from a process. The system call execlp() replaces the executing process by a new process image which executes the application specified as its parameter. Arguments can also be specified.

In simple words execlp() is a function that takes single or multiple strings and a Null pointer as parameter. First string defines the path of an executable file that we want to be executed and then the next strings are the commands that we want to run.

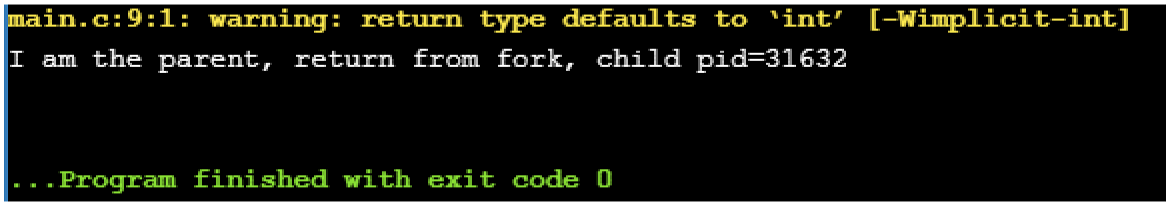
Execlp("Path/Directory", "Command1",...,NULL)

The given program first runs the else statement and because of fork it runs second time and till that time it moves in child class and the pid will have 0 value in it. The else if statement becomes true so it reaches execlp(). Execlp() is moving to path '/bin/ls' and executing the command 'ls' which is listing all the files.CODE:



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

// Required by for routine
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>

main()
{
    int pid;
    pid = fork();

     if (pid < 0)
    { // error occurred
        fprintf(stderr, "Fork failed!\n");
        exit(-1);
    }
    else if (pid == 0)
    { // child process
        printf("I am the child, return from fork=%d\n", pid);
        execlp("/bin/ls", "ls", NULL);
    }
    else
    { // parent process
        printf("I am the parent, return from fork, child pid=%d\n", pid);
    }
}
```

OUTPUT:



```
main.c:9:1: warning: return type defaults to 'int' [-Wimplicit-int]
I am the parent, return from fork, child pid=31632


...Program finished with exit code 0
```

BAHRIA UNIVERSITY

Karachi Campus

LAB EXPERIMENT NO. 7

LIST OF TASKS

TASK NO	OBJECTIVE
Task 1	Implement the codes of Example#01 and Example#02. Discuss what you have learned after implementing and executing these two codes. Provide snapshots of the generated outputs.
Task 2	The FCFS source code given above assumes that the arrival times for processes are provided in ascending order of time. Modify the code in C language for random process arrival times.
Task 3	Implement Round Robin Algorithm in C language
Task 4	Implement SPN algorithm in C.

Submitted On:

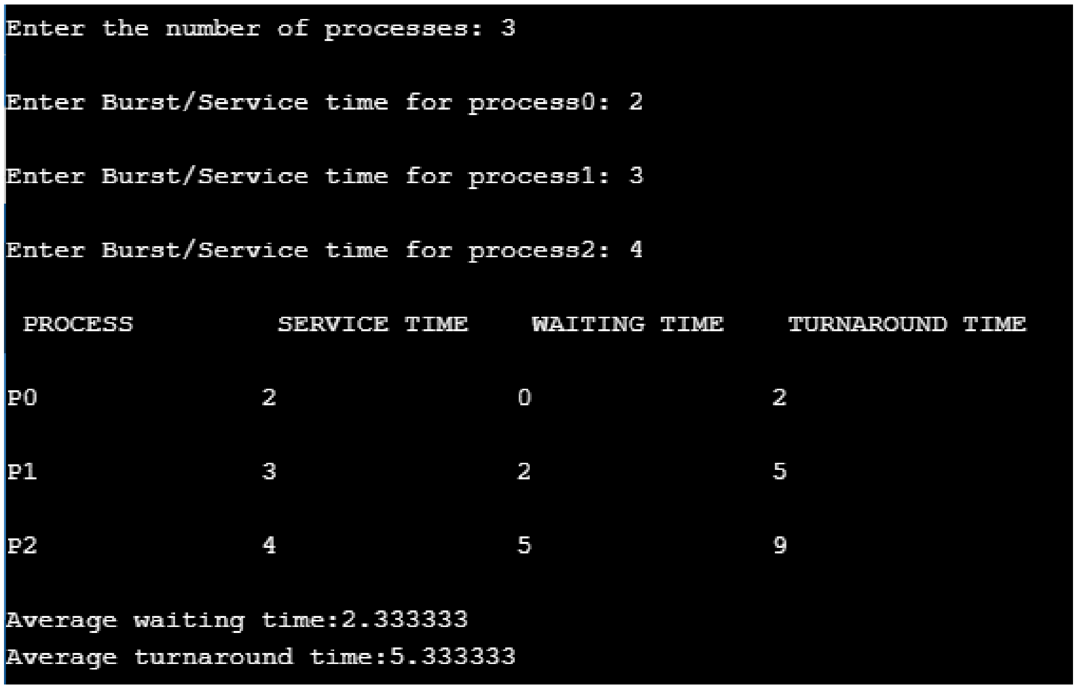
19/3/2020

(Date: DD/MM/YY)

Task 1

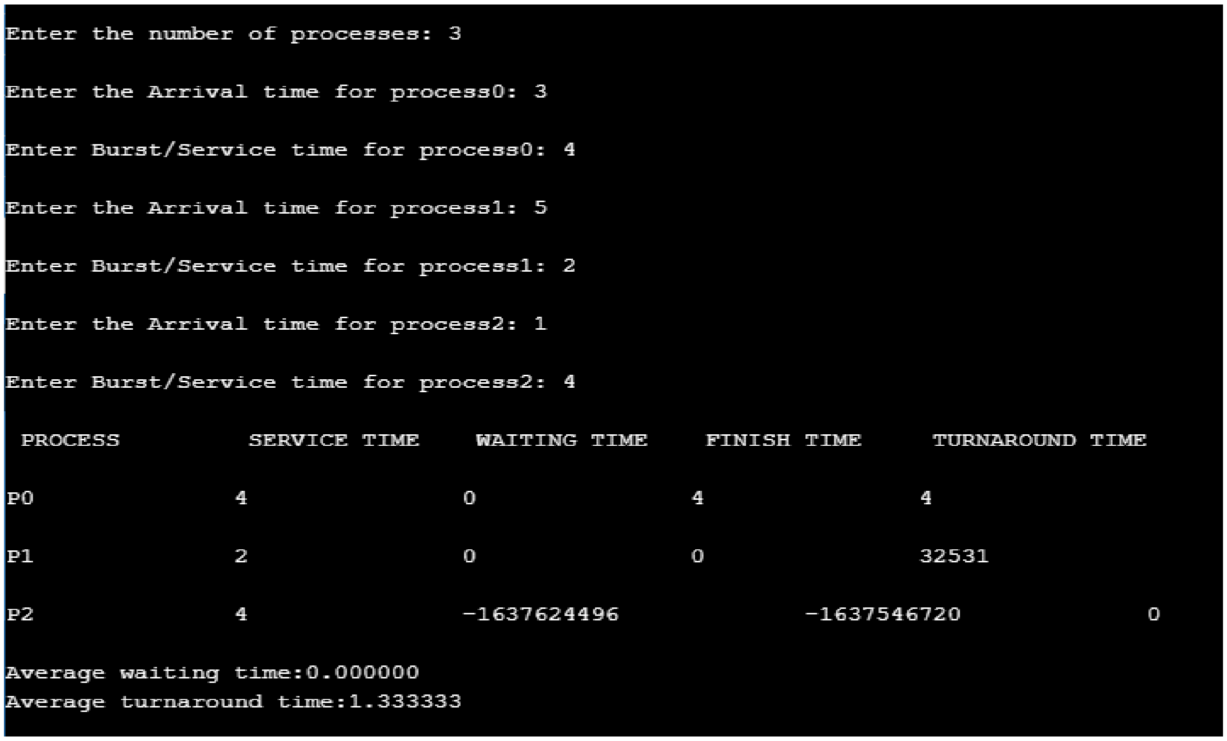
Implement the codes of Example#01 and Example#02. Discuss what you have learned after implementing and executing these two codes. Provide snapshots of the generated outputs.

OUTPUT (Example 1):



```
Enter the number of processes: 3

Enter Burst/Service time for process0: 2

Enter Burst/Service time for process1: 3

Enter Burst/Service time for process2: 4

 PROCESS          SERVICE TIME    WAITING TIME    TURNAROUND TIME

P0               2               0               2

P1               3               2               5

P2               4               5               9

Average waiting time:2.333333
Average turnaround time:5.333333
```

OUTPUT (Example 2):

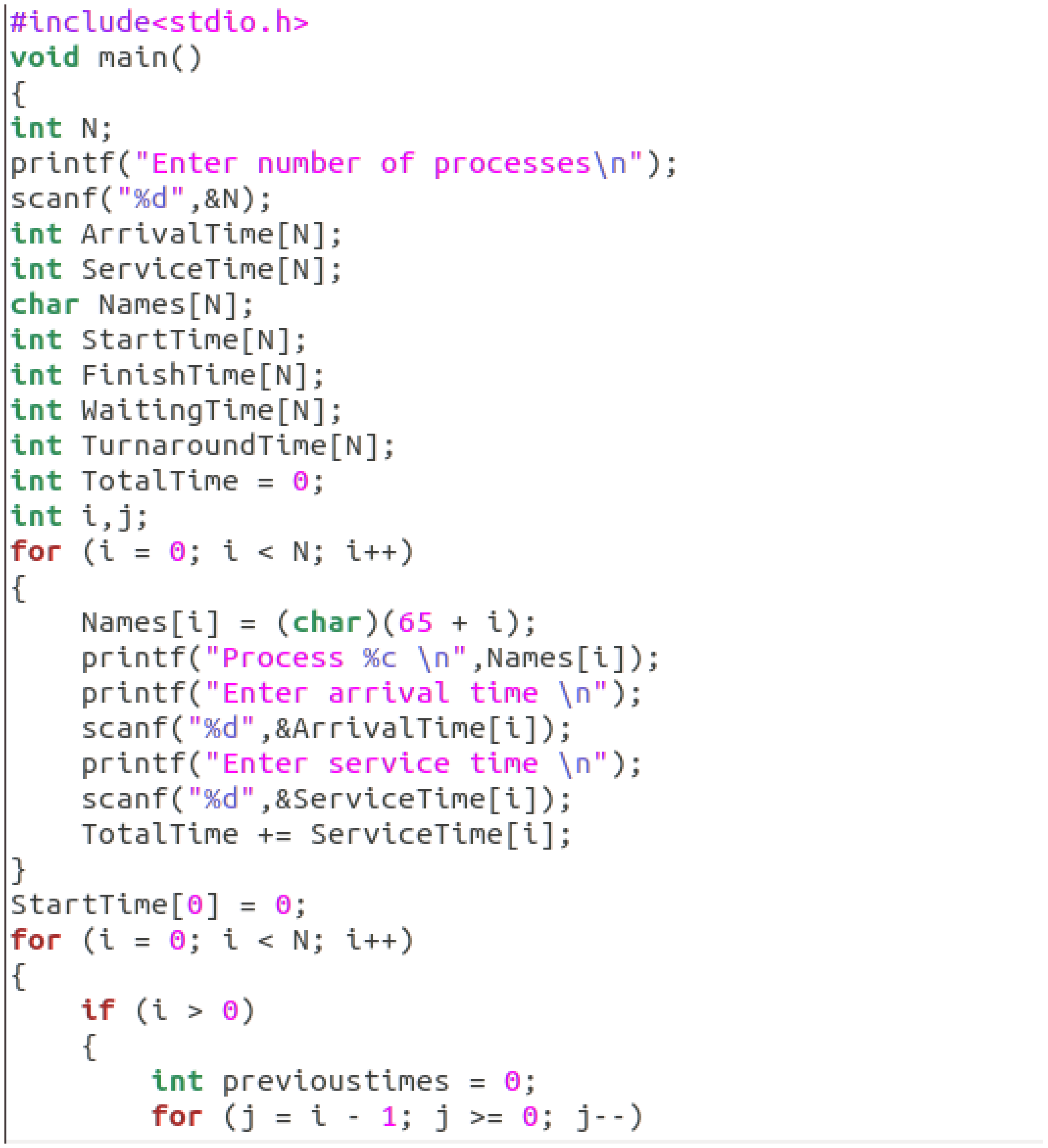


```
Enter the number of processes: 3

Enter the Arrival time for process0: 3

Enter Burst/Service time for process0: 4

Enter the Arrival time for process1: 5

Enter Burst/Service time for process1: 2

Enter the Arrival time for process2: 1

Enter Burst/Service time for process2: 4

 PROCESS          SERVICE TIME    WAITING TIME    FINISH TIME     TURNAROUND TIME

P0               4               0               4               4

P1               2               0               0               32531

P2               4               -1637624496             -1637546720             0

Average waiting time:0.000000
Average turnaround time:1.333333
```

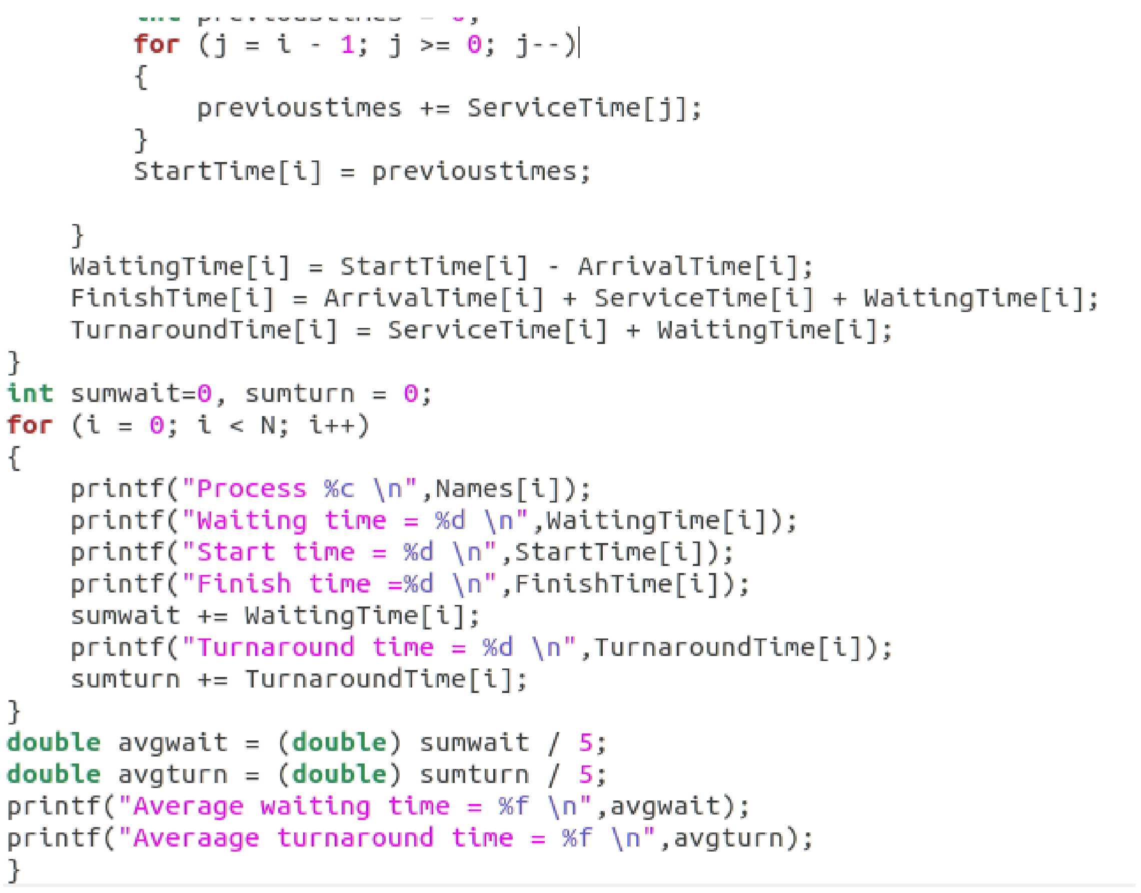
Task 2

The FCFS source code given above assumes that the arrival times for processes are provided in ascending order of time. Modify the code in C language for random process arrival times.

Code:

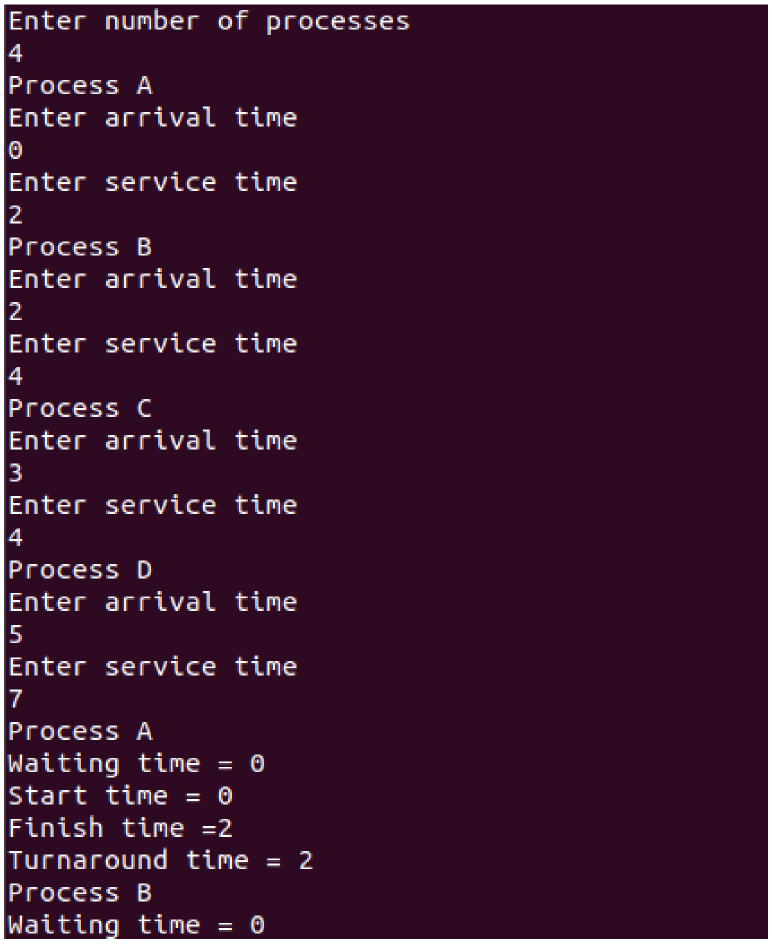


```c
#include<stdio.h>
void main()
{
int N;
printf("Enter number of processes\n");
scanf("%d",&N);
int ArrivalTime[N];
int ServiceTime[N];
char Names[N];
int StartTime[N];
int FinishTime[N];
int WaitingTime[N];
int TurnaroundTime[N];
int TotalTime = 0;
int i,j;
for (i = 0; i < N; i++)
{
    Names[i] = (char)(65 + i);
    printf("Process %c \n",Names[i]);
    printf("Enter arrival time \n");
    scanf("%d",&ArrivalTime[i]);
    printf("Enter service time \n");
    scanf("%d",&ServiceTime[i]);
    TotalTime += ServiceTime[i];
}
StartTime[0] = 0;
for (i = 0; i < N; i++)
{
    if (i > 0)
    {
        int previoustimes = 0;
        for (j = i - 1; j >= 0; j--)
```

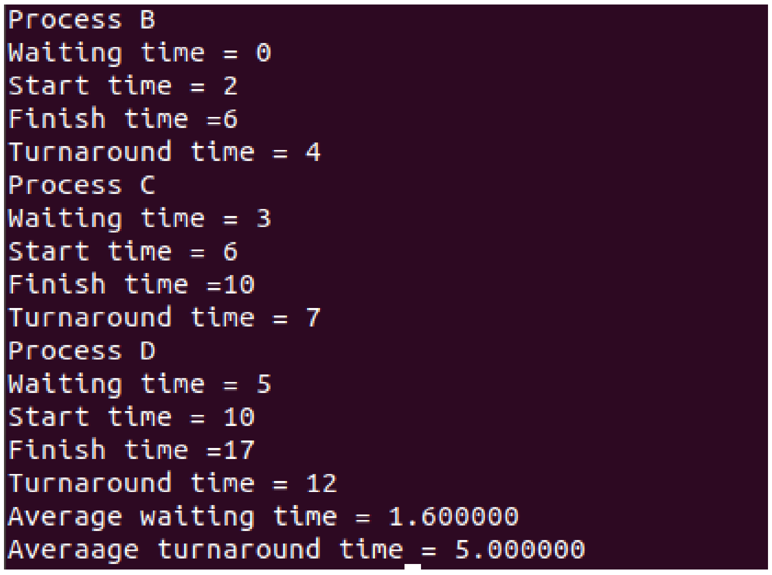


```
        int previoustimes = 0;
        for (j = i - 1; j >= 0; j--)
        {
            previoustimes += ServiceTime[j];
        }
        StartTime[i] = previoustimes;

    }
    WaitingTime[i] = StartTime[i] - ArrivalTime[i];
    FinishTime[i] = ArrivalTime[i] + ServiceTime[i] + WaitingTime[i];
    TurnaroundTime[i] = ServiceTime[i] + WaitingTime[i];
}
int sumwait=0, sumturn = 0;
for (i = 0; i < N; i++)
{
    printf("Process %c \n",Names[i]);
    printf("Waiting time = %d \n",WaitingTime[i]);
    printf("Start time = %d \n",StartTime[i]);
    printf("Finish time =%d \n",FinishTime[i]);
    sumwait += WaitingTime[i];
    printf("Turnaround time = %d \n",TurnaroundTime[i]);
    sumturn += TurnaroundTime[i];
}
double avgwait = (double) sumwait / 5;
double avgturn = (double) sumturn / 5;
printf("Average waiting time = %f \n",avgwait);
printf("Averaage turnaround time = %f \n",avgturn);
}
```

Output:



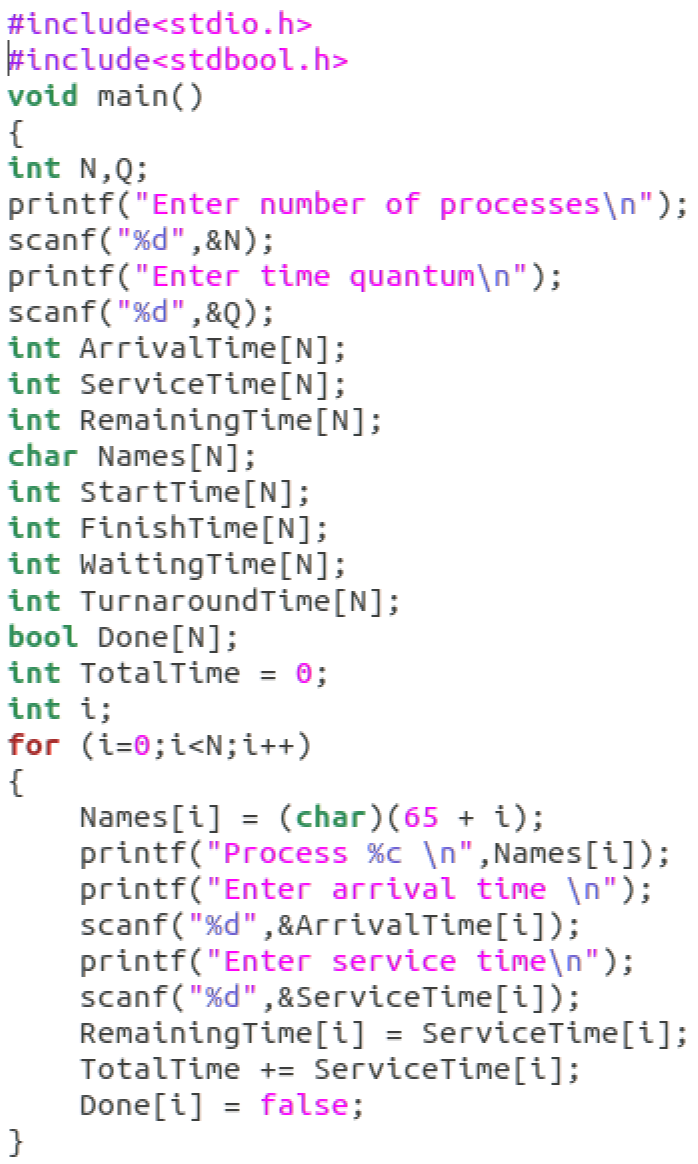
```
Enter number of processes
4
Process A
Enter arrival time
0
Enter service time
2
Process B
Enter arrival time
2
Enter service time
4
Process C
Enter arrival time
3
Enter service time
4
Process D
Enter arrival time
5
Enter service time
7
Process A
Waiting time = 0
Start time = 0
Finish time =2
Turnaround time = 2
Process B
Waiting time = 0
```



```
Process B
Waiting time = 0
Start time = 2
Finish time =6
Turnaround time = 4
Process C
Waiting time = 3
Start time = 6
Finish time =10
Turnaround time = 7
Process D
Waiting time = 5
Start time = 10
Finish time =17
Turnaround time = 12
Average waiting time = 1.600000
Averaage turnaround time = 5.000000
```

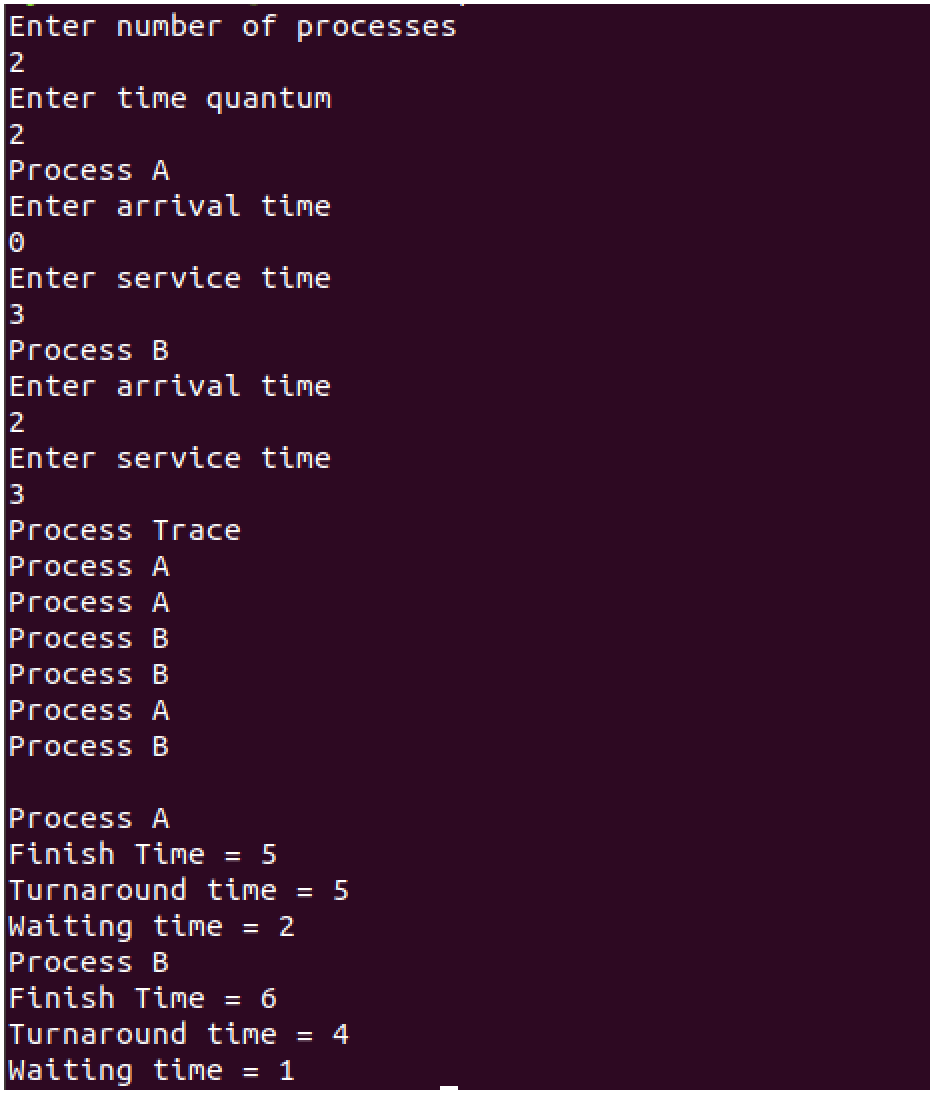
Task 3

Implement Round Robin Algorithm in C language.



```c
#include<stdio.h>
#include<stdbool.h>
void main()
{
int N,Q;
printf("Enter number of processes\n");
scanf("%d",&N);
printf("Enter time quantum\n");
scanf("%d",&Q);
int ArrivalTime[N];
int ServiceTime[N];
int RemainingTime[N];
char Names[N];
int StartTime[N];
int FinishTime[N];
int WaitingTime[N];
int TurnaroundTime[N];
bool Done[N];
int TotalTime = 0;
int i;
for (i=0;i<N;i++)
{
    Names[i] = (char)(65 + i);
    printf("Process %c \n",Names[i]);
    printf("Enter arrival time \n");
    scanf("%d",&ArrivalTime[i]);
    printf("Enter service time\n");
    scanf("%d",&ServiceTime[i]);
    RemainingTime[i] = ServiceTime[i];
    TotalTime += ServiceTime[i];
    Done[i] = false;
}
```

OUTPUT:

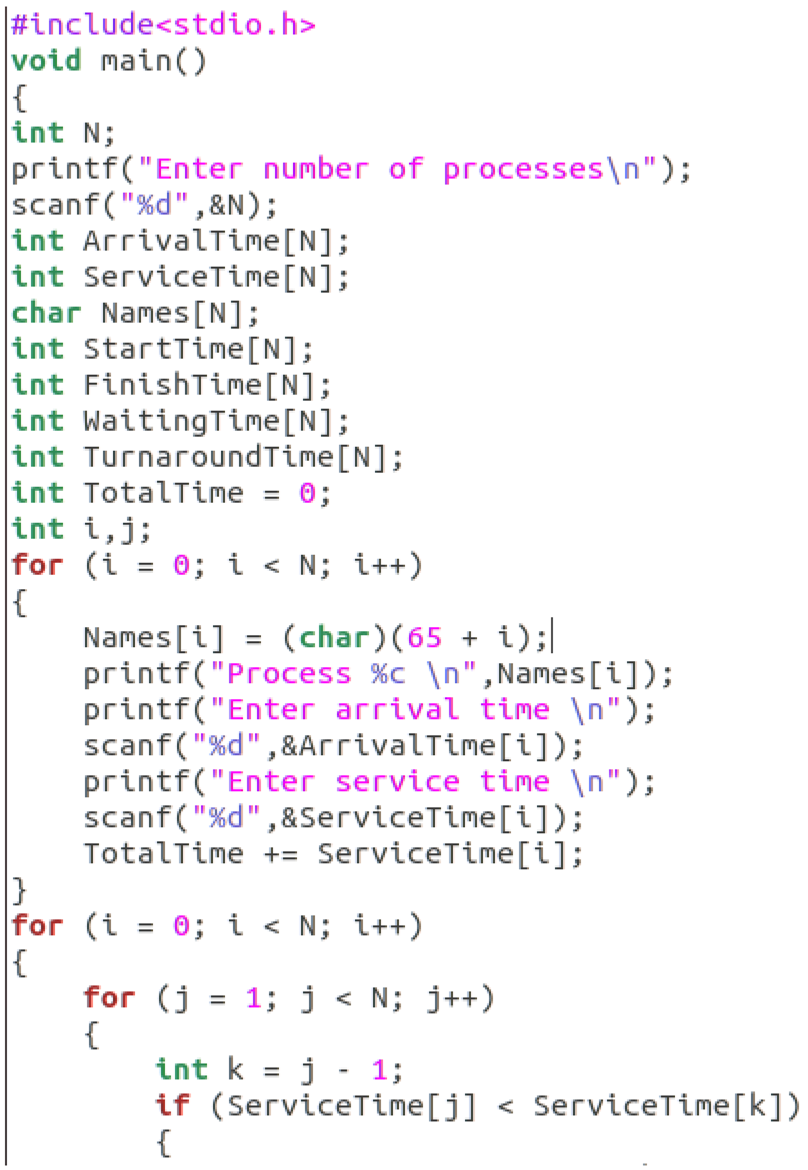


```
Enter number of processes
2
Enter time quantum
2
Process A
Enter arrival time
0
Enter service time
3
Process B
Enter arrival time
2
Enter service time
3
Process Trace
Process A
Process A
Process B
Process B
Process A
Process B

Process A
Finish Time = 5
Turnaround time = 5
Waiting time = 2
Process B
Finish Time = 6
Turnaround time = 4
Waiting time = 1
```

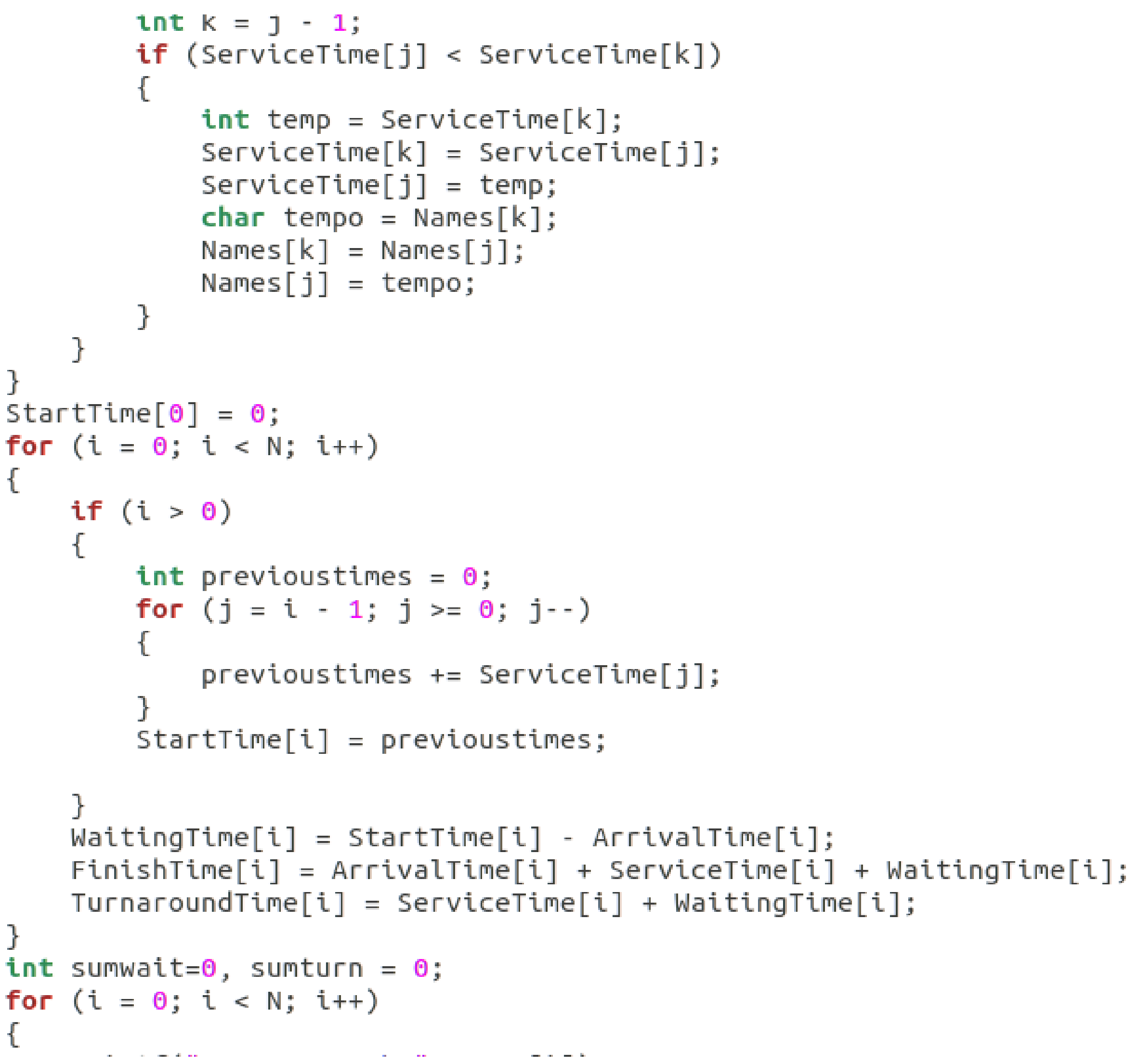
Task 4

Implement SPN algorithm in C.

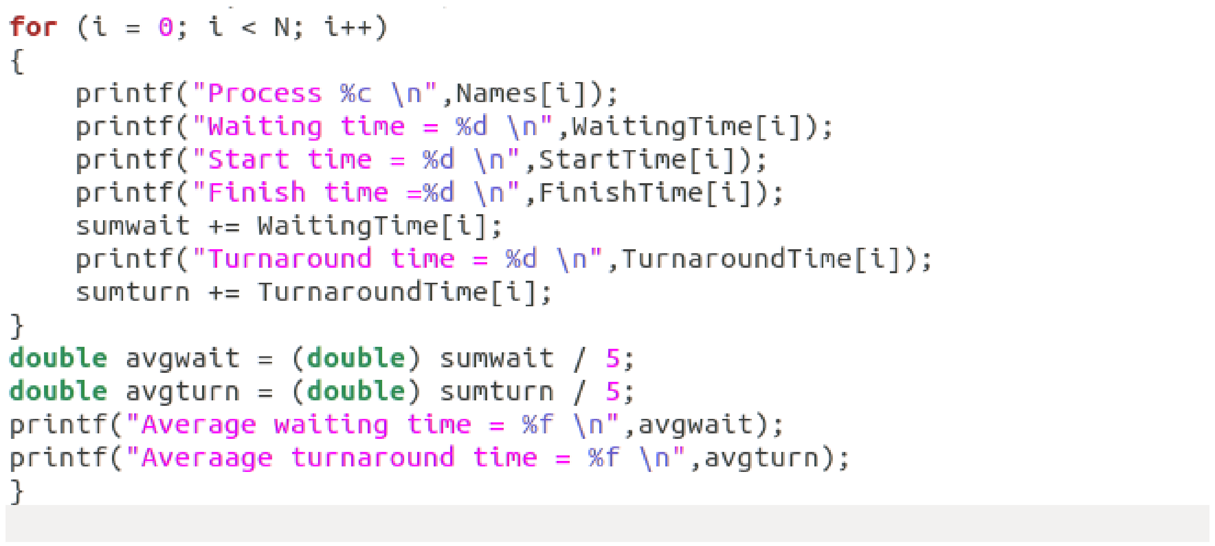
Code:



```c
#include<stdio.h>
void main()
{
int N;
printf("Enter number of processes\n");
scanf("%d",&N);
int ArrivalTime[N];
int ServiceTime[N];
char Names[N];
int StartTime[N];
int FinishTime[N];
int WaitingTime[N];
int TurnaroundTime[N];
int TotalTime = 0;
int i,j;
for (i = 0; i < N; i++)
{
    Names[i] = (char)(65 + i);
    printf("Process %c \n",Names[i]);
    printf("Enter arrival time \n");
    scanf("%d",&ArrivalTime[i]);
    printf("Enter service time \n");
    scanf("%d",&ServiceTime[i]);
    TotalTime += ServiceTime[i];
}
for (i = 0; i < N; i++)
{
    for (j = 1; j < N; j++)
    {
        int k = j - 1;
        if (ServiceTime[j] < ServiceTime[k])
        {
```

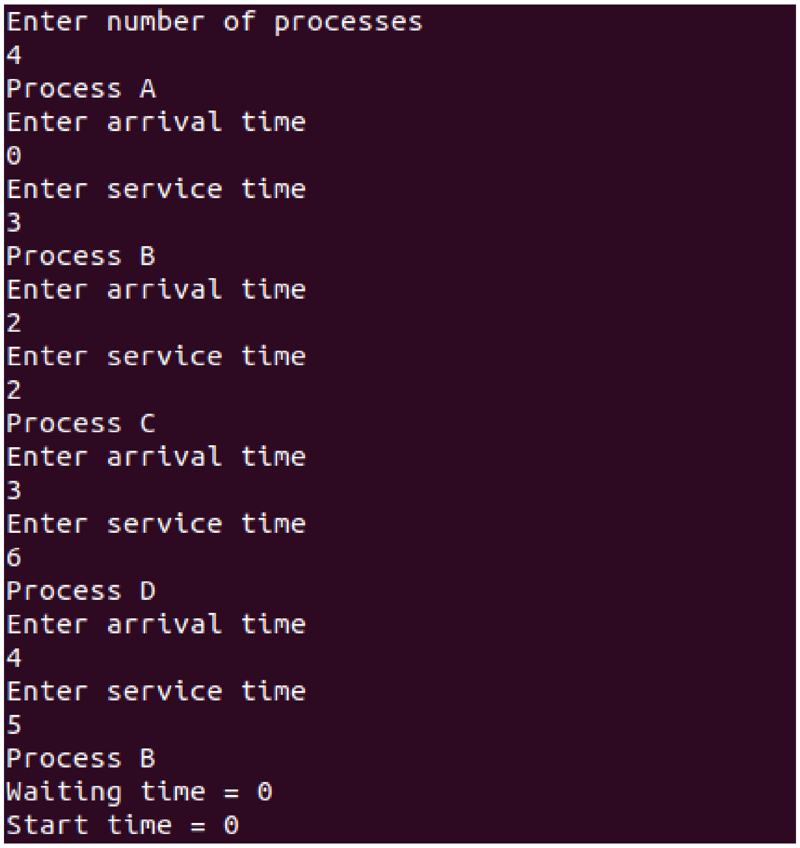


```
        int k = j - 1;
        if (ServiceTime[j] < ServiceTime[k])
        {
            int temp = ServiceTime[k];
            ServiceTime[k] = ServiceTime[j];
            ServiceTime[j] = temp;
            char tempo = Names[k];
            Names[k] = Names[j];
            Names[j] = tempo;
        }
    }
}
StartTime[0] = 0;
for (i = 0; i < N; i++)
{
    if (i > 0)
    {
        int previoustimes = 0;
        for (j = i - 1; j >= 0; j--)
        {
            previoustimes += ServiceTime[j];
        }
        StartTime[i] = previoustimes;

    }
    WaitingTime[i] = StartTime[i] - ArrivalTime[i];
    FinishTime[i] = ArrivalTime[i] + ServiceTime[i] + WaitingTime[i];
    TurnaroundTime[i] = ServiceTime[i] + WaitingTime[i];
}
int sumwait=0, sumturn = 0;
for (i = 0; i < N; i++)
{
```

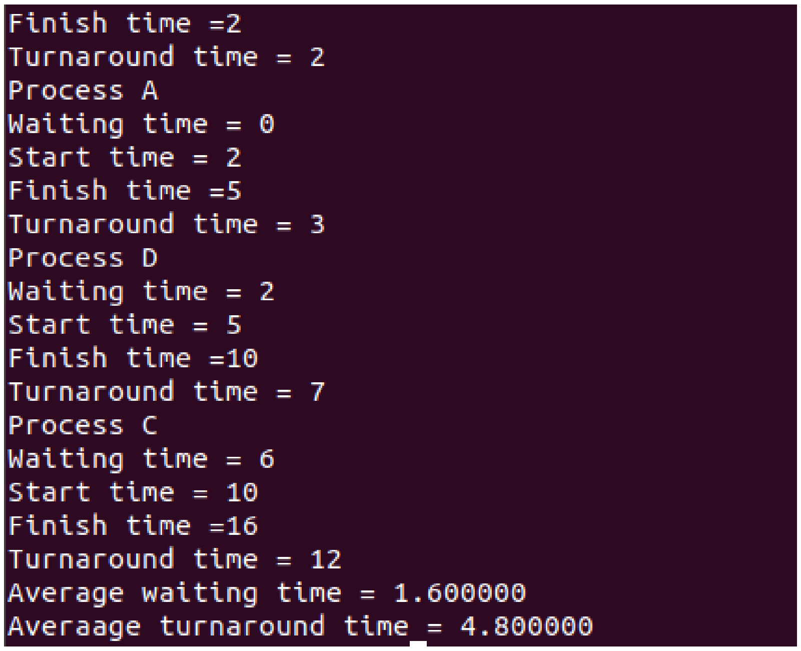


```
for (i = 0; i < N; i++)
{
    printf("Process %c \n",Names[i]);
    printf("Waiting time = %d \n",WaitingTime[i]);
    printf("Start time = %d \n",StartTime[i]);
    printf("Finish time =%d \n",FinishTime[i]);
    sumwait += WaitingTime[i];
    printf("Turnaround time = %d \n",TurnaroundTime[i]);
    sumturn += TurnaroundTime[i];
}
double avgwait = (double) sumwait / 5;
double avgturn = (double) sumturn / 5;
printf("Average waiting time = %f \n",avgwait);
printf("Averaage turnaround time = %f \n",avgturn);
}
```

Output:



```
Enter number of processes
4
Process A
Enter arrival time
0
Enter service time
3
Process B
Enter arrival time
2
Enter service time
2
Process C
Enter arrival time
3
Enter service time
6
Process D
Enter arrival time
4
Enter service time
5
Process B
Waiting time = 0
Start time = 0
```



```
Finish time =2
Turnaround time = 2
Process A
Waiting time = 0
Start time = 2
Finish time =5
Turnaround time = 3
Process D
Waiting time = 2
Start time = 5
Finish time =10
Turnaround time = 7
Process C
Waiting time = 6
Start time = 10
Finish time =16
Turnaround time = 12
Average waiting time = 1.600000
Averaage turnaround time = 4.800000
```

BAHRIA UNIVERSITY

Karachi Campus

LAB EXPERIMENT NO.8

LIST OF TASKS

TASK NO	OBJECTIVE
Task 1	Execute both commands. Try different options and observe the results. What have you learnt? Discuss.
Task 2	Explore HTOP, including its all options. Attach Outputs for the same. Discuss your Observations.
Task 3	Write a multithreaded C program for performing summation of numbers

Submitted On:

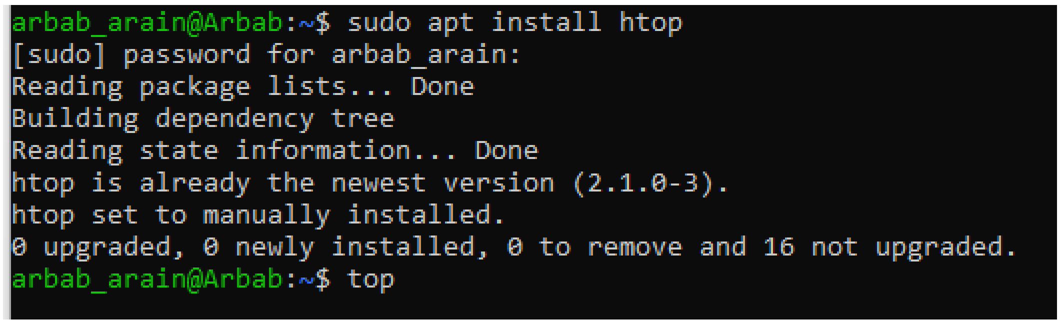
26/3/2020

(Date: DD/MM/YY)

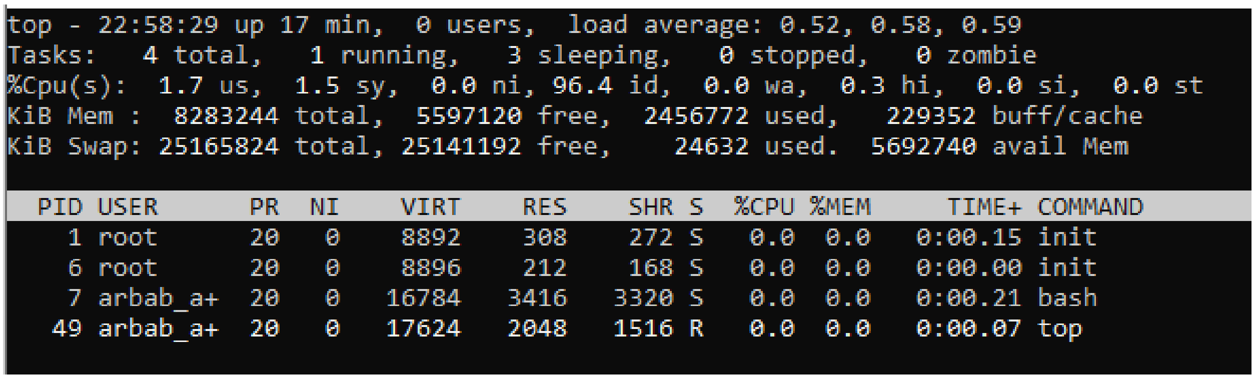
Task1: Execute both commands. Try different options and observe the results. What have you learnt? Discuss.

TOP command:

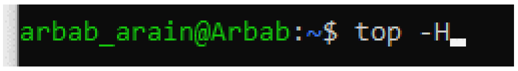
Top command is used to display Linux processes and all information related to current processes and threads that are running or being managed by the kernel. All the information is user configurable. This program provides limited interactive interface for process manipulation as well as a much more extensive interface for personal configuration.



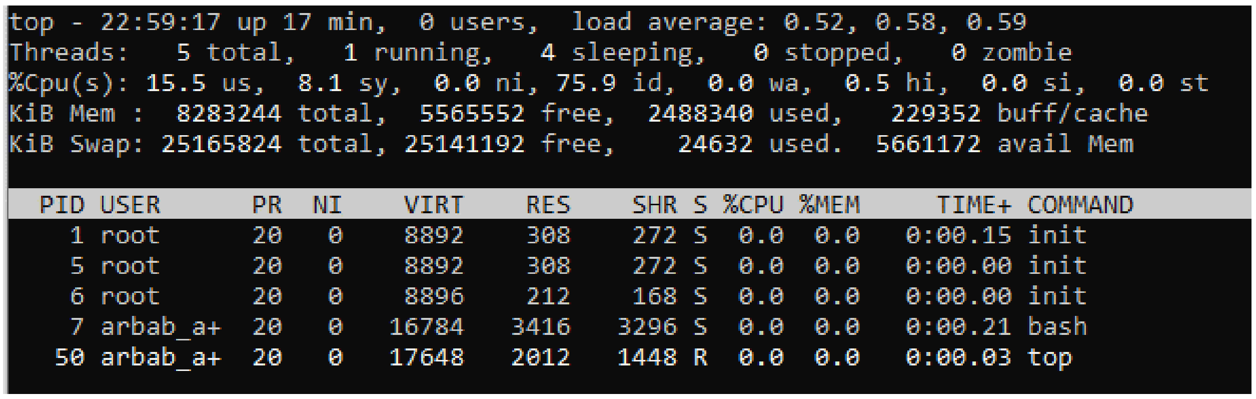
```
arbab_arain@Arbab:~$ sudo apt install htop
[sudo] password for arbab_arain:
Reading package lists... Done
Building dependency tree
Reading state information... Done
htop is already the newest version (2.1.0-3).
htop set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 16 not upgraded.
arbab_arain@Arbab:~$ top
```



```
top - 22:58:29 up 17 min,  0 users,  load average: 0.52, 0.58, 0.59
Tasks:   4 total,   1 running,   3 sleeping,   0 stopped,   0 zombie
%Cpu(s):  1.7 us,  1.5 sy,  0.0 ni, 96.4 id,  0.0 wa,  0.3 hi,  0.0 si,  0.0 st
KiB Mem :  8283244 total,  5597120 free,  2456772 used,   229352 buff/cache
KiB Swap: 25165824 total, 25141192 free,    24632 used.  5692740 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
    1 root      20   0    8892    308    272 S   0.0  0.0   0:00.15 init
    6 root      20   0    8896    212    168 S   0.0  0.0   0:00.00 init
    7 arbab_a+  20   0   16784   3416   3320 S   0.0  0.0   0:00.21 bash
   49 arbab_a+  20   0   17624   2048   1516 R   0.0  0.0   0:00.07 top
```



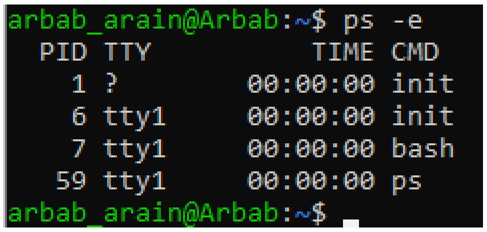
```
arbab_arain@Arbab:~$ top -H
```



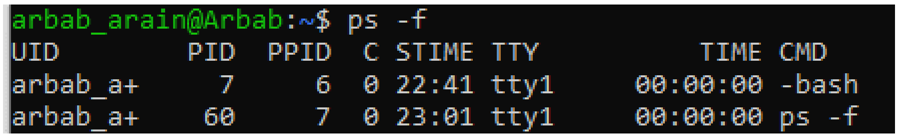
```
top - 22:59:17 up 17 min,  0 users,  load average: 0.52, 0.58, 0.59
Threads:   5 total,   1 running,   4 sleeping,   0 stopped,   0 zombie
%Cpu(s): 15.5 us,  8.1 sy,  0.0 ni, 75.9 id,  0.0 wa,  0.5 hi,  0.0 si,  0.0 st
KiB Mem :  8283244 total,  5565552 free,  2488340 used,   229352 buff/cache
KiB Swap: 25165824 total, 25141192 free,    24632 used.  5661172 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
    1 root      20   0    8892    308    272 S  0.0  0.0   0:00.15 init
    5 root      20   0    8892    308    272 S  0.0  0.0   0:00.00 init
    6 root      20   0    8896    212    168 S  0.0  0.0   0:00.00 init
    7 arbab_a+  20   0   16784   3416   3296 S  0.0  0.0   0:00.21 bash
   50 arbab_a+  20   0   17648   2012   1448 R  0.0  0.0   0:00.03 top
```

PS command:

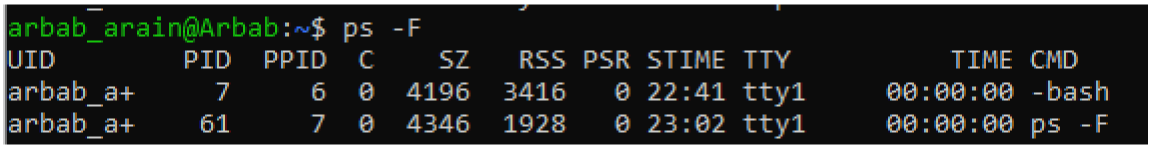
Ps command is used to report a snapshot of the current running processes. Ps displays information about a selection of the active processes. Ps selects all processes that are related to current user and associated with the same terminal as the process that invoked it.



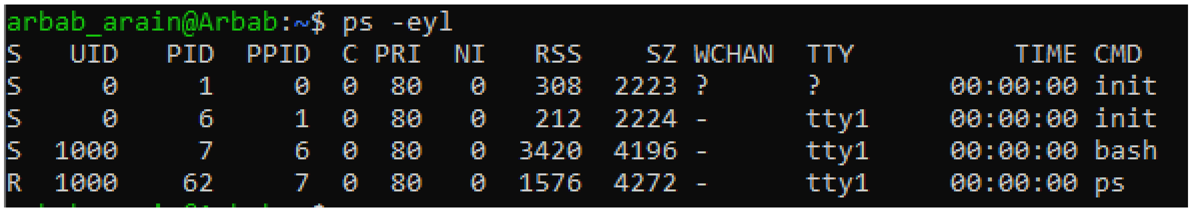
```
arbab_arain@Arbab:~$ ps -e
  PID TTY          TIME CMD
    1 ?        00:00:00 init
    6 tty1     00:00:00 init
    7 tty1     00:00:00 bash
   59 tty1     00:00:00 ps
arbab_arain@Arbab:~$
```



```
arbab_arain@Arbab:~$ ps -f
UID        PID  PPID  C STIME TTY          TIME CMD
arbab_a+     7     6  0 22:41 tty1     00:00:00 -bash
arbab_a+    60     7  0 23:01 tty1     00:00:00 ps -f
```



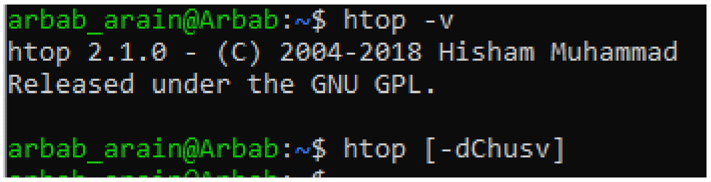
```
arbab_arain@Arbab:~$ ps -F
UID        PID  PPID  C    SZ   RSS PSR STIME TTY          TIME CMD
arbab_a+     7     6  0  4196  3416   0 22:41 tty1     00:00:00 -bash
arbab_a+    61     7  0  4346  1928   0 23:02 tty1     00:00:00 ps -F
```



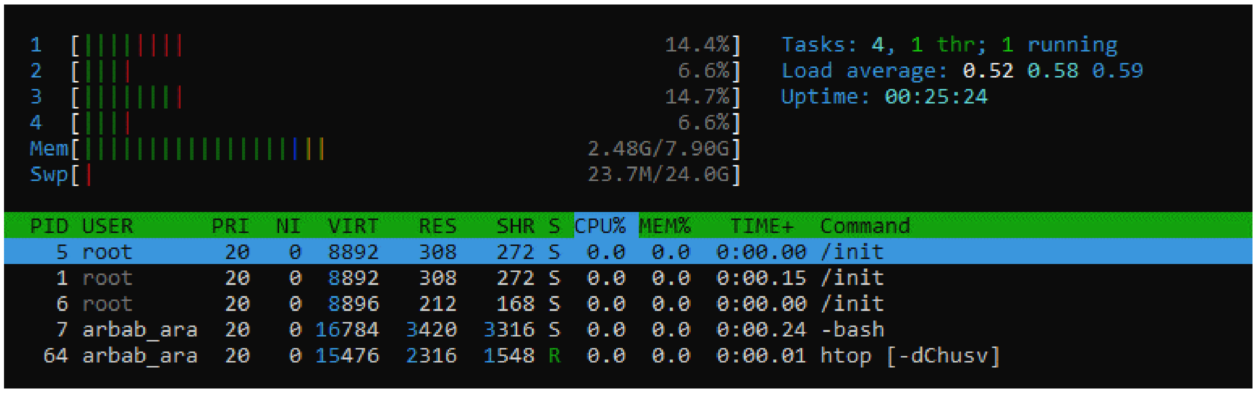
```
arbab_arain@Arbab:~$ ps -eyl
S   UID   PID  PPID  C PRI  NI   RSS    SZ WCHAN  TTY          TIME CMD
S     0     1     0  0  80   0   308  2223 ?      ?        00:00:00 init
S     0     6     1  0  80   0   212  2224 -      tty1     00:00:00 init
S  1000     7     6  0  80   0  3420  4196 -      tty1     00:00:00 bash
R  1000    62     7  0  80   0  1576  4272 -      tty1     00:00:00 ps
```

Task2: Explore HTOP, including its all options. Attach Outputs for the same. Discuss your Observations.

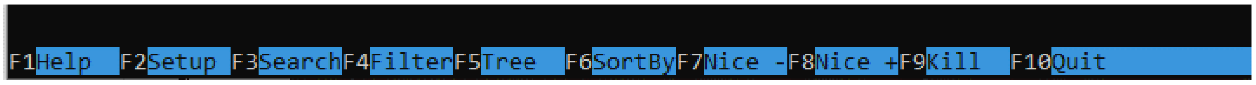
HTOP command in Linux System us a command line utility that allows the user to interactively monitor the system's vital resources or server's processes in real time.



```
arbab_arain@Arbab:~$ htop -v
htop 2.1.0 - (C) 2004-2018 Hisham Muhammad
Released under the GNU GPL.

arbab_arain@Arbab:~$ htop [-dChusv]
```



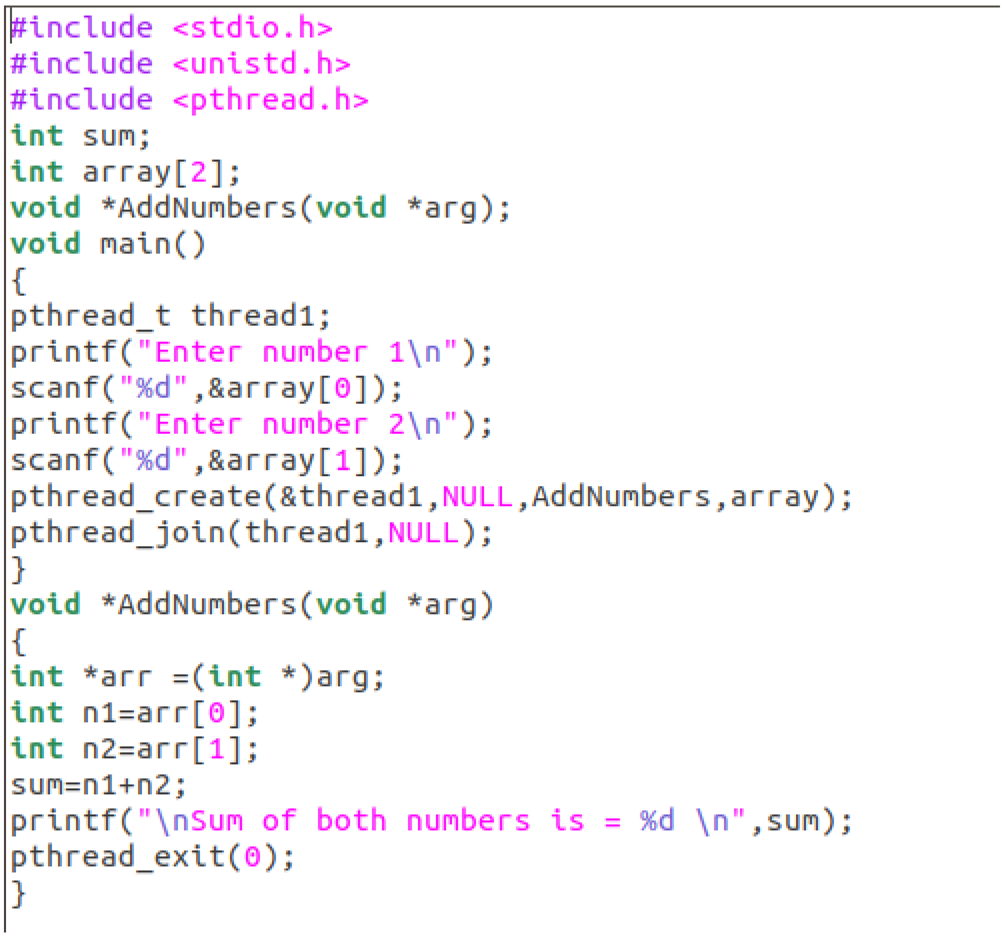
```
  1  [||||||||                                          14.4%]   Tasks: 4, 1 thr; 1 running
  2  [||||                                               6.6%]   Load average: 0.52 0.58 0.59
  3  [||||||||                                          14.7%]   Uptime: 00:25:24
  4  [||||                                               6.6%]
  Mem[||||||||||||||||||                           2.48G/7.90G]
  Swp[|                                            23.7M/24.0G]

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
    5 root       20   0  8892   308   272 S  0.0  0.0  0:00.00 /init
    1 root       20   0  8892   308   272 S  0.0  0.0  0:00.15 /init
    6 root       20   0  8896   212   168 S  0.0  0.0  0:00.00 /init
    7 arbab_ara  20   0 16784  3420  3316 S  0.0  0.0  0:00.24 -bash
   64 arbab_ara  20   0 15476  2316  1548 R  0.0  0.0  0:00.01 htop [-dChusv]
```



```
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

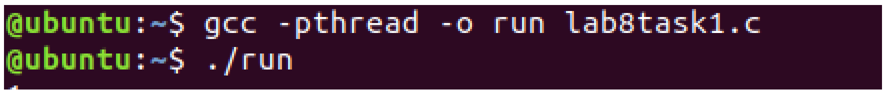
Task3: Write a multithreaded C program for performing summation of numbers

Code:

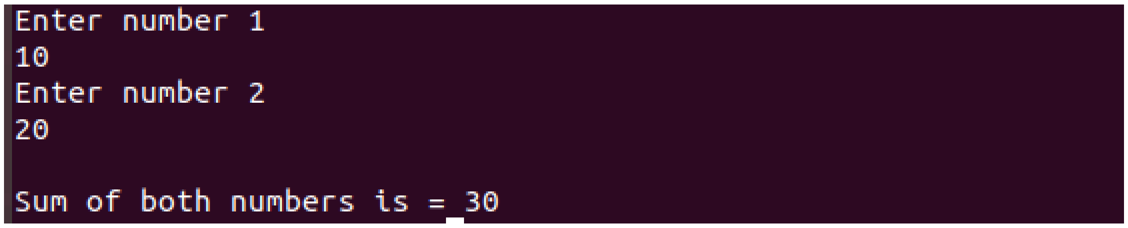


```c
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>
int sum;
int array[2];
void *AddNumbers(void *arg);
void main()
{
pthread_t thread1;
printf("Enter number 1\n");
scanf("%d",&array[0]);
printf("Enter number 2\n");
scanf("%d",&array[1]);
pthread_create(&thread1,NULL,AddNumbers,array);
pthread_join(thread1,NULL);
}
void *AddNumbers(void *arg)
{
int *arr =(int *)arg;
int n1=arr[0];
int n2=arr[1];
sum=n1+n2;
printf("\nSum of both numbers is = %d \n",sum);
pthread_exit(0);
}
```

Output:



```
@ubuntu:~$ gcc -pthread -o run lab8task1.c
@ubuntu:~$ ./run
```



```
Enter number 1
10
Enter number 2
20

Sum of both numbers is = 30
```

BAHRIA UNIVERSITY

Karachi Campus

LAB EXPERIMENT NO. 9

LIST OF TASKS

TASK NO	OBJECTIVE
Task 1	Semaphore is one of the concurrency mechanisms available. Find out about more concurrency mechanisms. How do these mechanisms protect critical sections? Compare their implementations with wait () and signal () operations of semaphores.
Task 2	Implement Producer Consumer Problem in C language using Semaphore.

Submitted On:

16/4/2020

(Date: DD/MM/YY)

Task1:

Semaphore is one of the concurrency mechanisms available. Find out about more concurrency mechanisms. How do these mechanisms protect critical sections? Compare their implementations with wait() and signal() operations of semaphores.

Answer:

A semaphore does the allows multiple number of threads to access a particular resource until and unless there is a need of mutual exclusion.

A lock allows only one thread to enter the part that's locked and the lock is not shared with any other processes.

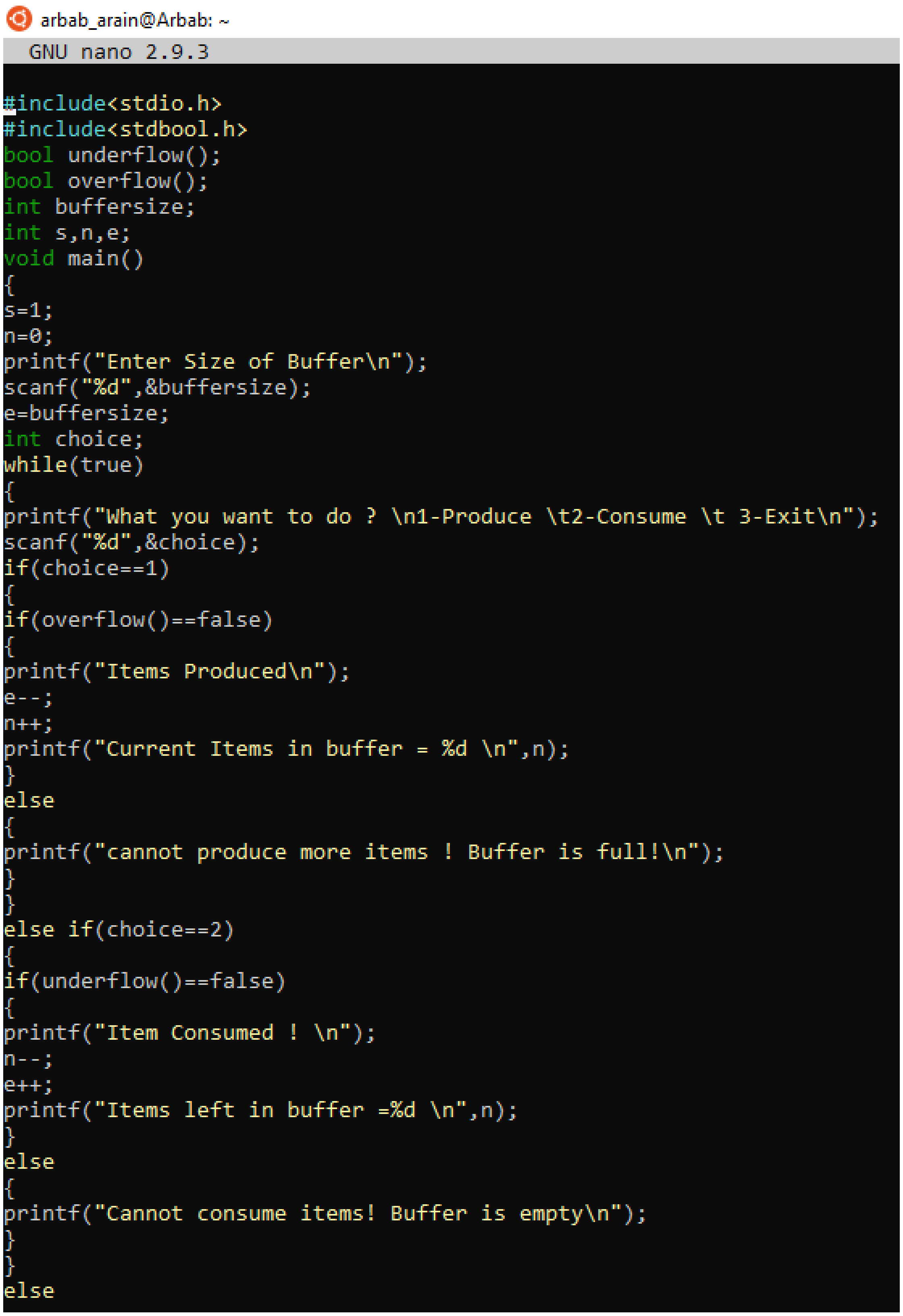
A mutex is the same as a lock but it can be shared by multiple processes. In mutex, acquire will let first caller through, and then block next until release

In condition synchronization, acquiring will block first caller until release.

Task2:

Implement Producer Consumer Problem in C language using Semaphore.

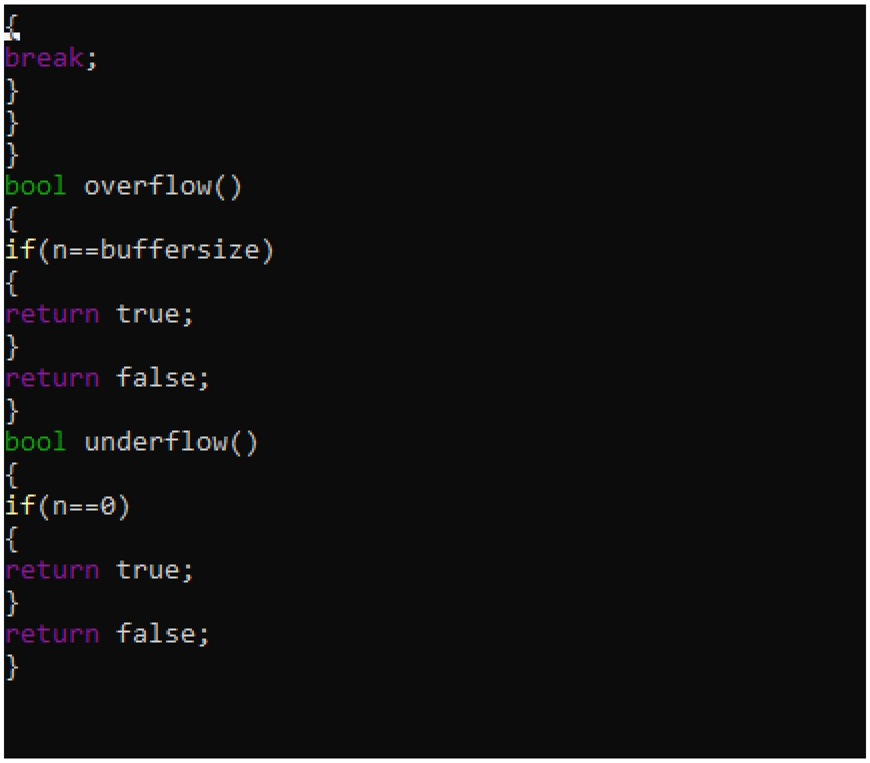
Code:



arbab_arain@Arbab: ~

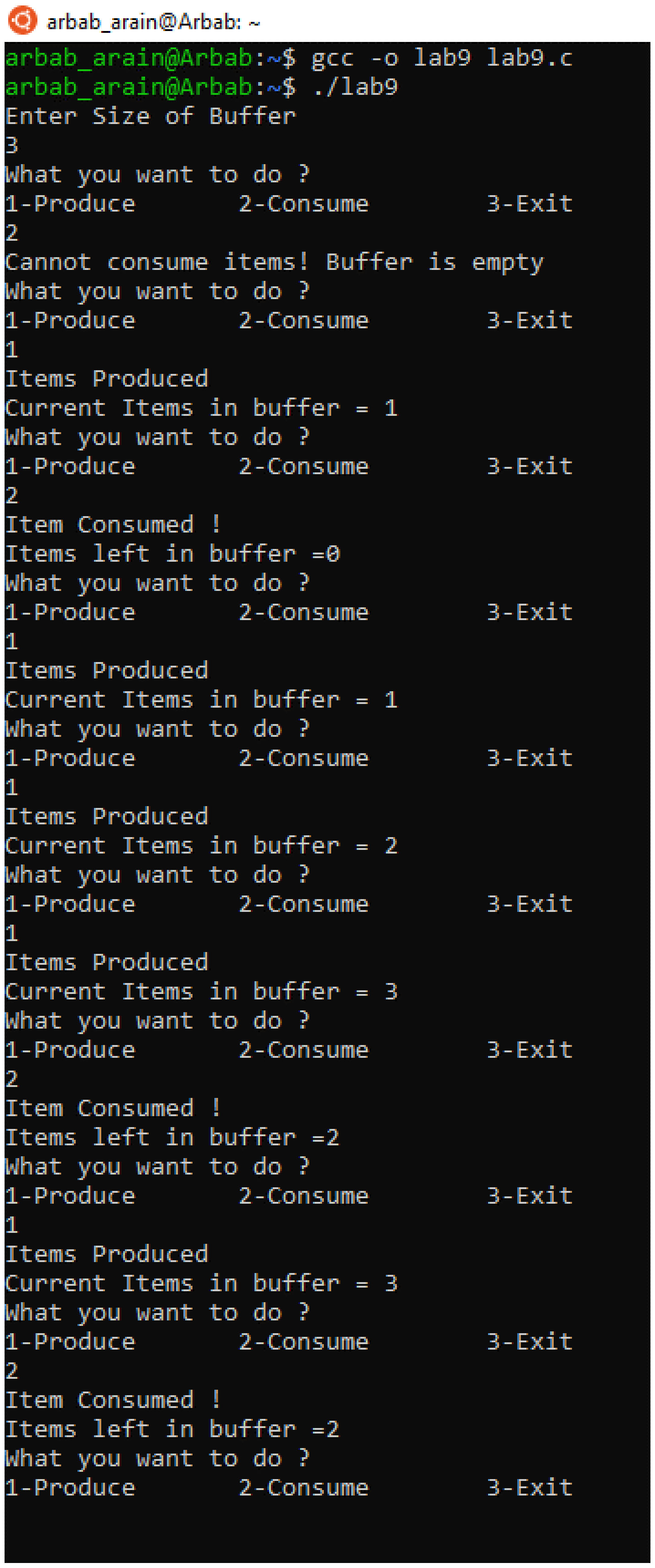
GNU nano 2.9.3

```
#include<stdio.h>
#include<stdbool.h>
bool underflow();
bool overflow();
int buffersize;
int s,n,e;
void main()
{
s=1;
n=0;
printf("Enter Size of Buffer\n");
scanf("%d",&buffersize);
e=buffersize;
int choice;
while(true)
{
printf("What you want to do ? \n1-Produce \t2-Consume \t 3-Exit\n");
scanf("%d",&choice);
if(choice==1)
{
if(overflow()==false)
{
printf("Items Produced\n");
e--;
n++;
printf("Current Items in buffer = %d \n",n);
}
else
{
printf("cannot produce more items ! Buffer is full!\n");
}
}
else if(choice==2)
{
if(underflow()==false)
{
printf("Item Consumed ! \n");
n--;
e++;
printf("Items left in buffer =%d \n",n);
}
else
{
printf("Cannot consume items! Buffer is empty\n");
}
}
else
```



```
{
break;
}
}
}
bool overflow()
{
if(n==buffersize)
{
return true;
}
return false;
}
bool underflow()
{
if(n==0)
{
return true;
}
return false;
}
```

Output:


```
arbab_arain@Arbab: ~
arbab_arain@Arbab:~$ gcc -o lab9 lab9.c
arbab_arain@Arbab:~$ ./lab9
Enter Size of Buffer
3
What you want to do ?
1-Produce       2-Consume       3-Exit
2
Cannot consume items! Buffer is empty
What you want to do ?
1-Produce       2-Consume       3-Exit
1
Items Produced
Current Items in buffer = 1
What you want to do ?
1-Produce       2-Consume       3-Exit
2
Item Consumed !
Items left in buffer =0
What you want to do ?
1-Produce       2-Consume       3-Exit
1
Items Produced
Current Items in buffer = 1
What you want to do ?
1-Produce       2-Consume       3-Exit
1
Items Produced
Current Items in buffer = 2
What you want to do ?
1-Produce       2-Consume       3-Exit
1
Items Produced
Current Items in buffer = 3
What you want to do ?
1-Produce       2-Consume       3-Exit
2
Item Consumed !
Items left in buffer =2
What you want to do ?
1-Produce       2-Consume       3-Exit
1
Items Produced
Current Items in buffer = 3
What you want to do ?
1-Produce       2-Consume       3-Exit
2
Item Consumed !
Items left in buffer =2
What you want to do ?
1-Produce       2-Consume       3-Exit
```

BAHRIA UNIVERSITY

Karachi Campus

LAB EXPERIMENT NO. 10

LIST OF TASKS

TASK NO	OBJECTIVE
Task 1	Implement the Banker's Algorithm explained above in C language.
Task 2	Study and implement Safety algorithm and Resource-Request Algorithm in the above program.

Submitted On:

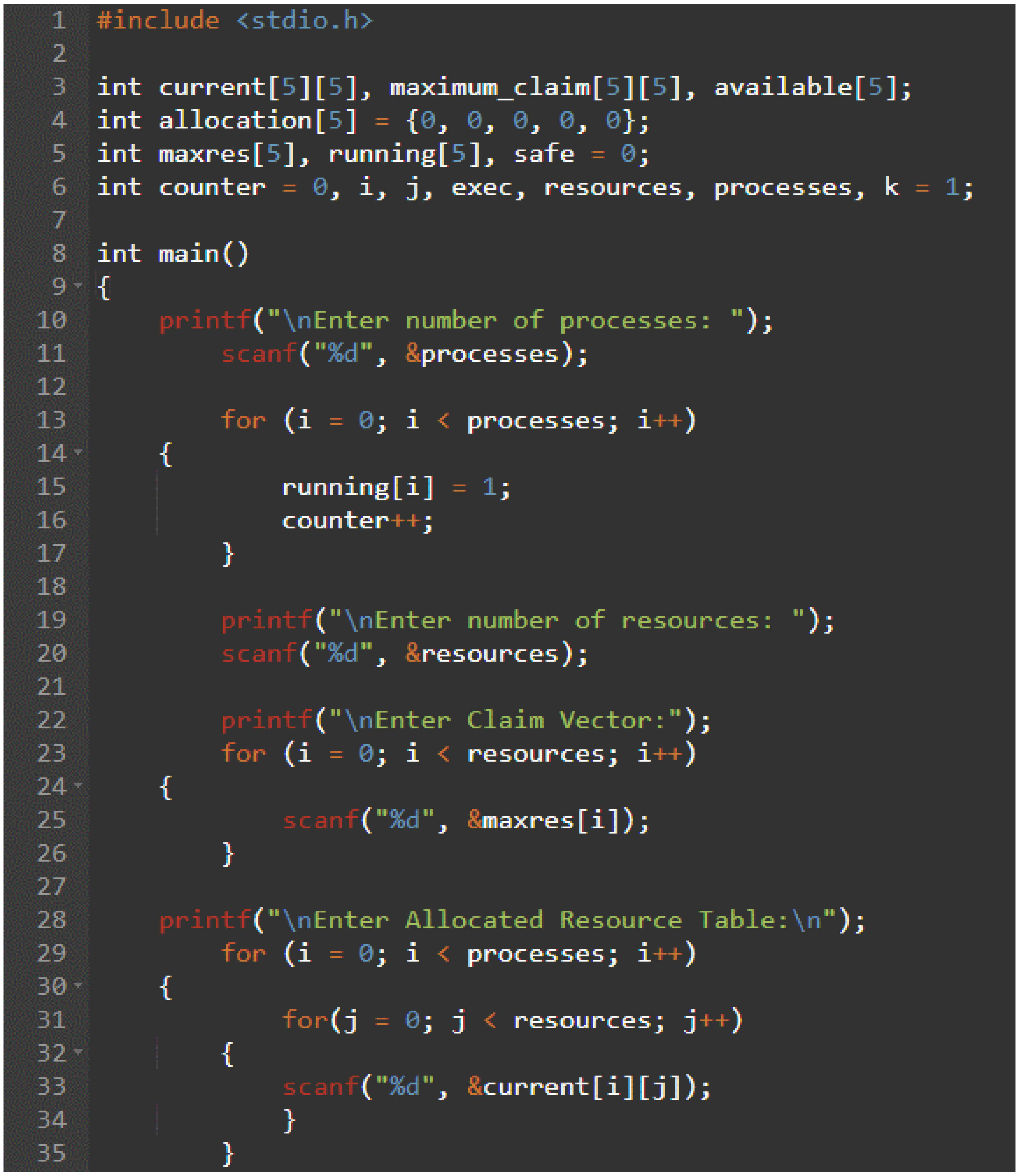
23/4/2020

(Date: DD/MM/YY)

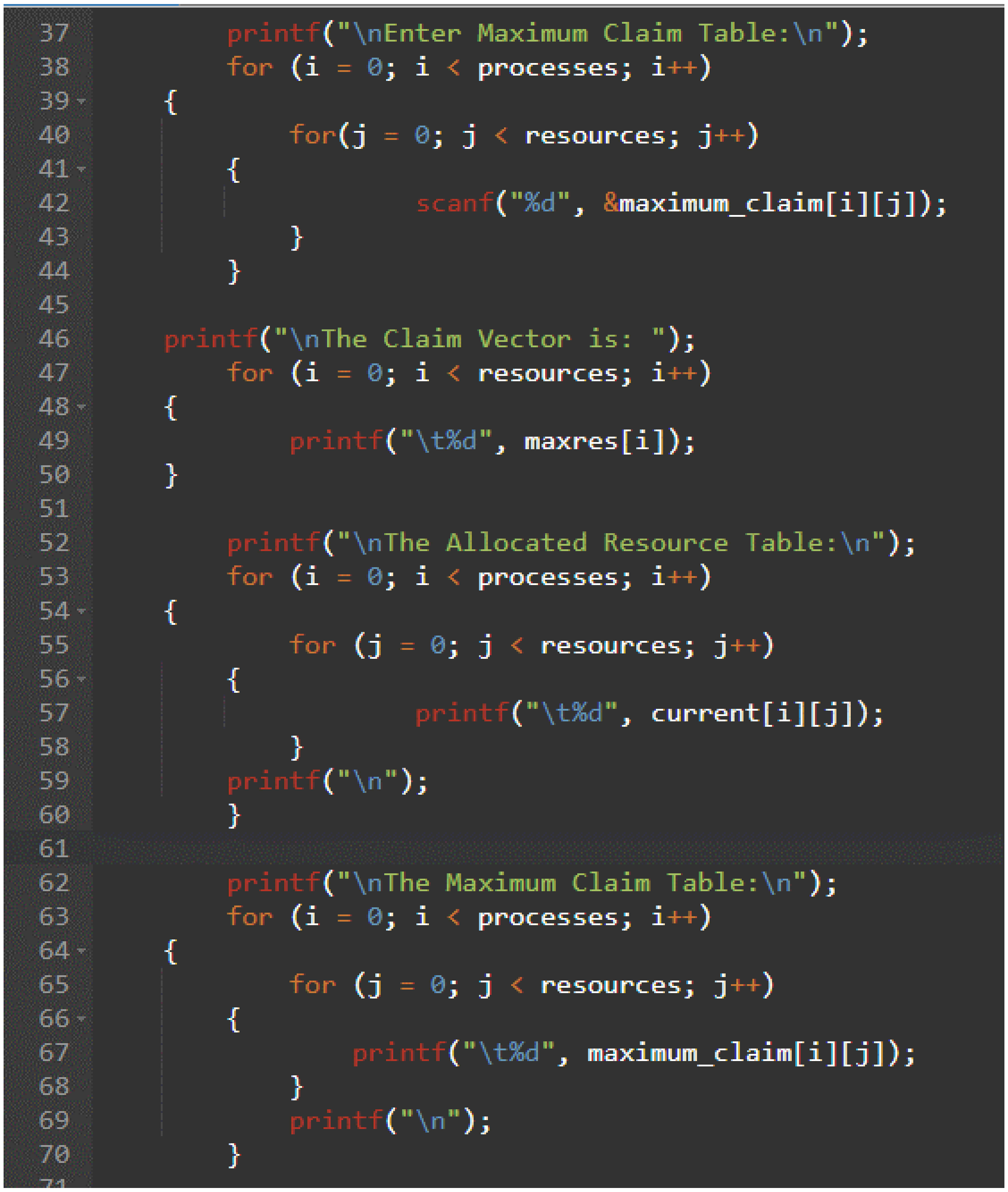
Task # 1:

Implement the Banker's Algorithm explained above in C language.

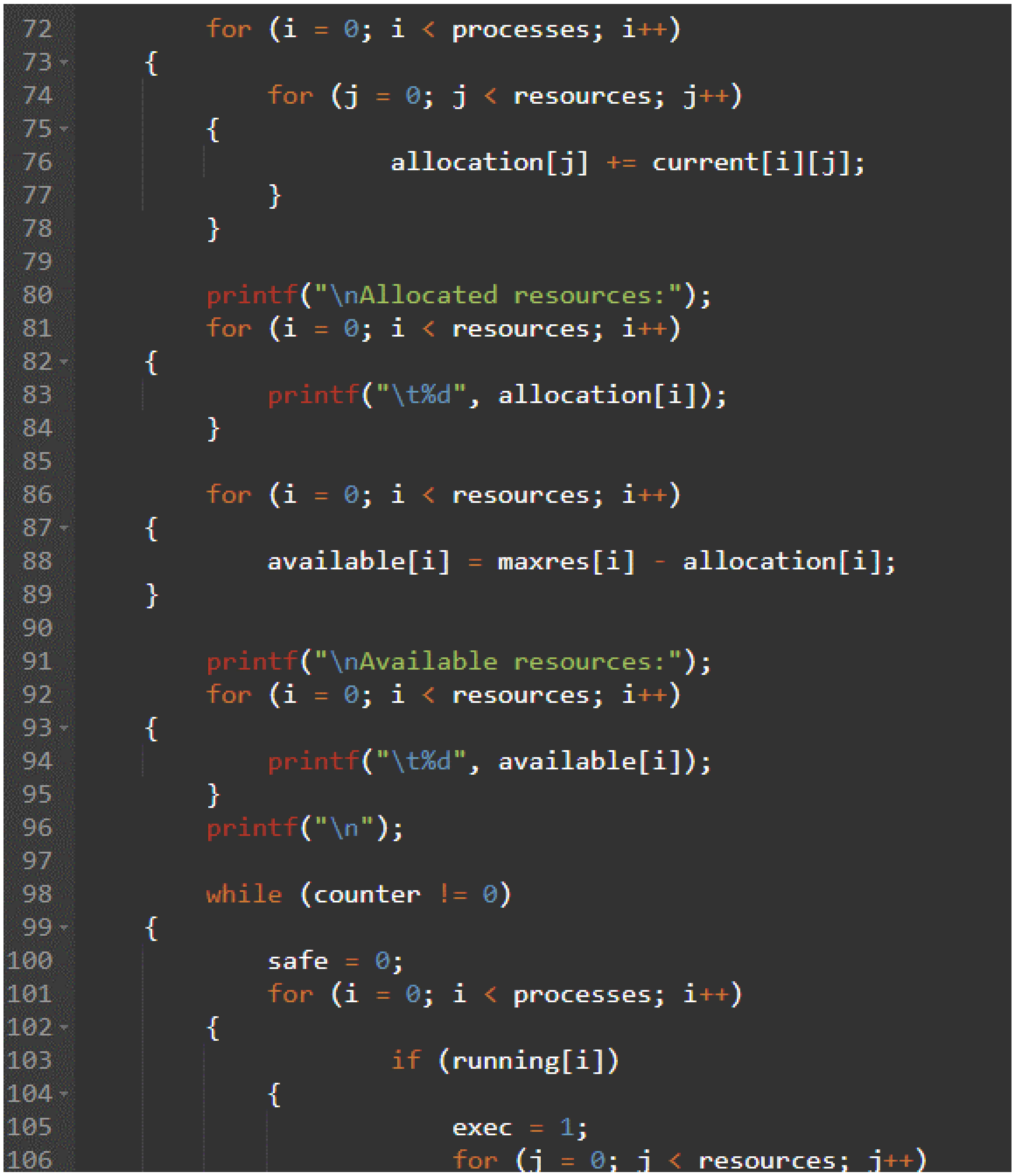
Code:



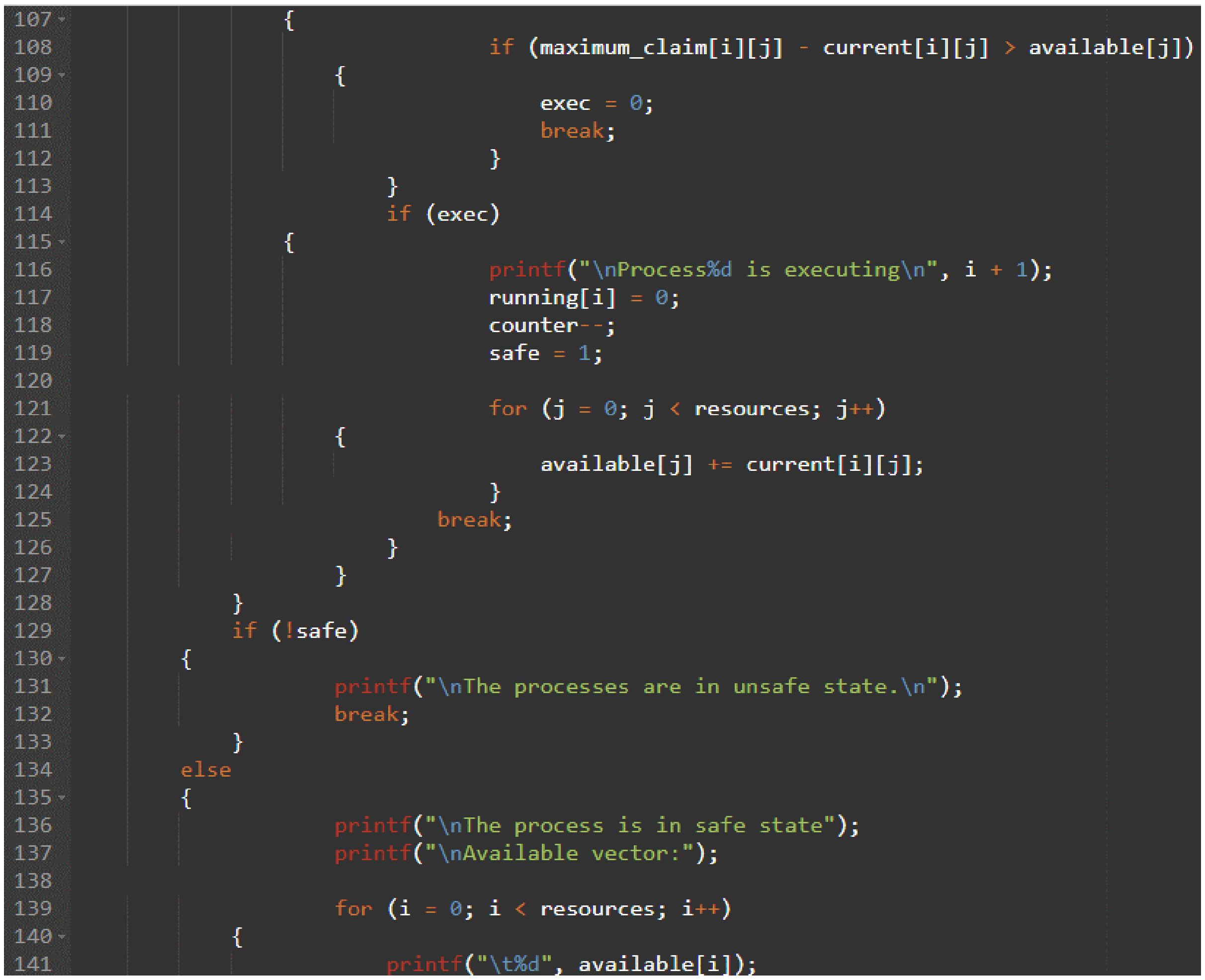
```c
#include <stdio.h>

int current[5][5], maximum_claim[5][5], available[5];
int allocation[5] = {0, 0, 0, 0, 0};
int maxres[5], running[5], safe = 0;
int counter = 0, i, j, exec, resources, processes, k = 1;

int main()
{
    printf("\nEnter number of processes: ");
        scanf("%d", &processes);

        for (i = 0; i < processes; i++)
    {
            running[i] = 1;
            counter++;
        }

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

        printf("\nEnter Claim Vector:");
        for (i = 0; i < resources; i++)
    {
            scanf("%d", &maxres[i]);
        }

    printf("\nEnter Allocated Resource Table:\n");
        for (i = 0; i < processes; i++)
    {
            for(j = 0; j < resources; j++)
        {
            scanf("%d", &current[i][j]);
            }
        }
```



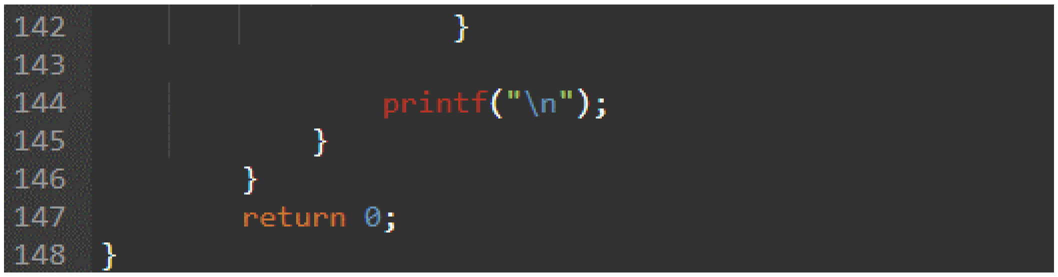
```
        printf("\nEnter Maximum Claim Table:\n");
        for (i = 0; i < processes; i++)
    {
            for(j = 0; j < resources; j++)
        {
                    scanf("%d", &maximum_claim[i][j]);
            }
        }

    printf("\nThe Claim Vector is: ");
        for (i = 0; i < resources; i++)
    {
            printf("\t%d", maxres[i]);
    }

        printf("\nThe Allocated Resource Table:\n");
        for (i = 0; i < processes; i++)
    {
            for (j = 0; j < resources; j++)
        {
                    printf("\t%d", current[i][j]);
            }
        printf("\n");
        }

        printf("\nThe Maximum Claim Table:\n");
        for (i = 0; i < processes; i++)
    {
            for (j = 0; j < resources; j++)
        {
                printf("\t%d", maximum_claim[i][j]);
            }
            printf("\n");
        }
```



```
        for (i = 0; i < processes; i++)
    {
            for (j = 0; j < resources; j++)
        {
                    allocation[j] += current[i][j];
            }
        }

        printf("\nAllocated resources:");
        for (i = 0; i < resources; i++)
    {
            printf("\t%d", allocation[i]);
        }

        for (i = 0; i < resources; i++)
    {
            available[i] = maxres[i] - allocation[i];
    }

        printf("\nAvailable resources:");
        for (i = 0; i < resources; i++)
    {
            printf("\t%d", available[i]);
        }
        printf("\n");

        while (counter != 0)
    {
            safe = 0;
            for (i = 0; i < processes; i++)
        {
                    if (running[i])
            {
                        exec = 1;
                        for (j = 0; j < resources; j++)
```

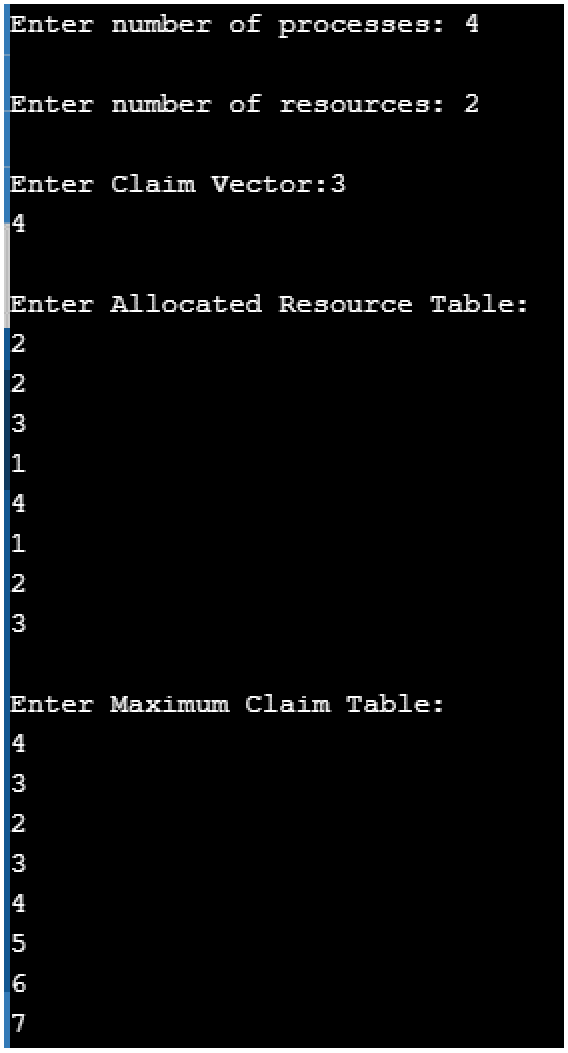


```c
                {
                                if (maximum_claim[i][j] - current[i][j] > available[j])
                    {
                                    exec = 0;
                                    break;
                                }
                        }
                        if (exec)
                {
                                printf("\nProcess%d is executing\n", i + 1);
                                running[i] = 0;
                                counter--;
                                safe = 1;

                                for (j = 0; j < resources; j++)
                    {
                                    available[j] += current[i][j];
                                }
                            break;
                        }
                    }
            }
            if (!safe)
        {
                    printf("\nThe processes are in unsafe state.\n");
                    break;
            }
        else
        {
                    printf("\nThe process is in safe state");
                    printf("\nAvailable vector:");

                    for (i = 0; i < resources; i++)
            {
                        printf("\t%d", available[i]);
```

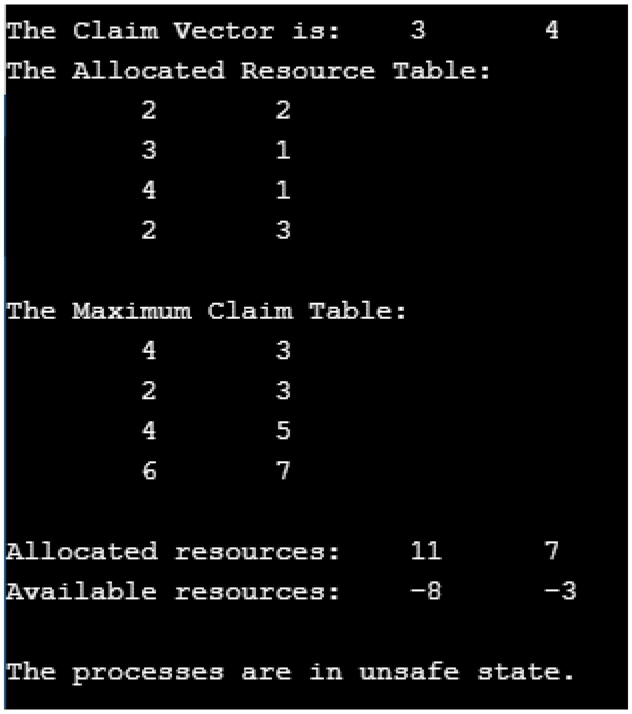


```c
                    }

                printf("\n");
            }
        }
        return 0;
}
```

Output:



```
Enter number of processes: 4

Enter number of resources: 2

Enter Claim Vector:3
4

Enter Allocated Resource Table:
2
2
3
1
4
1
2
3

Enter Maximum Claim Table:
4
3
2
3
4
5
6
7
```

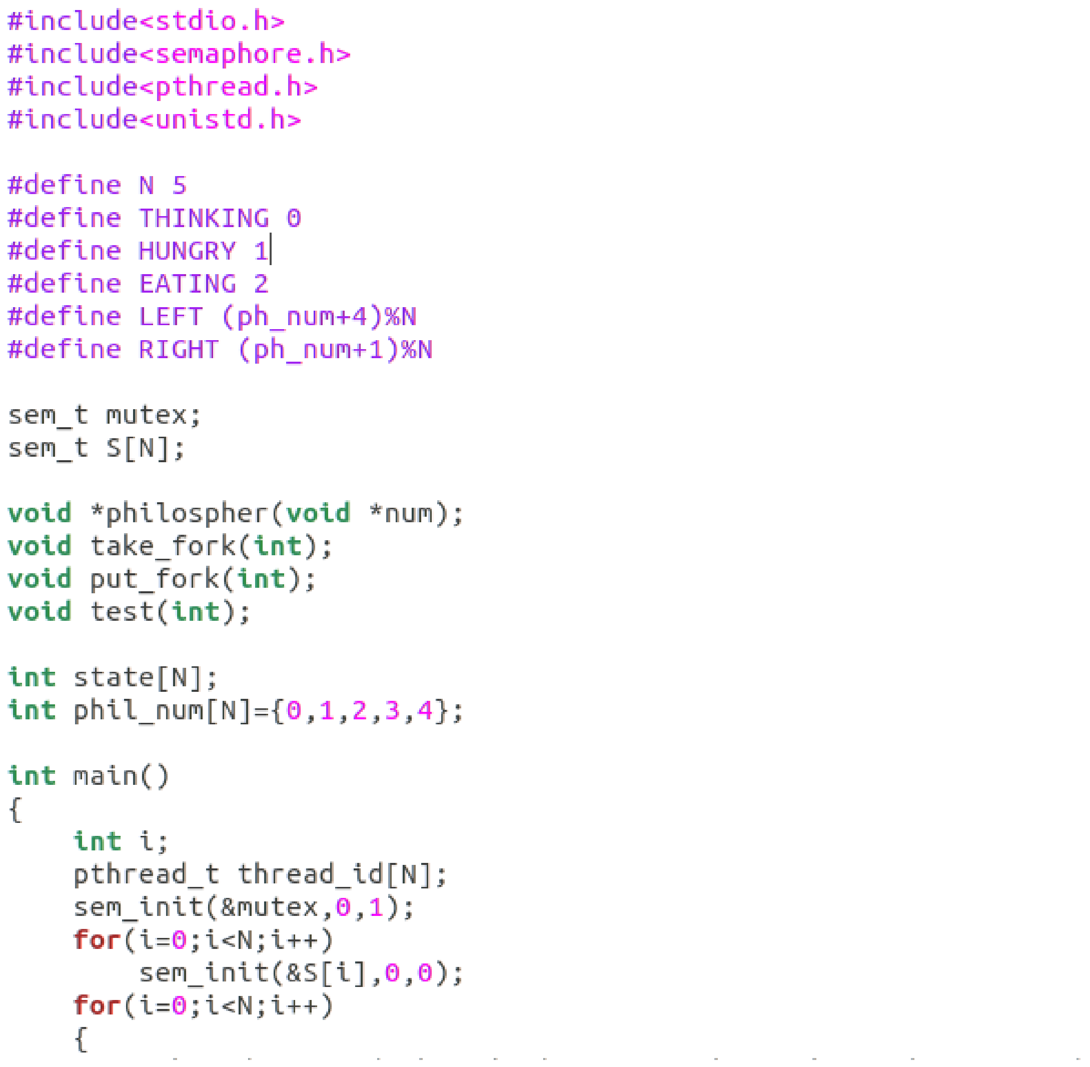


```
The Claim Vector is:    3       4
The Allocated Resource Table:
        2       2
        3       1
        4       1
        2       3

The Maximum Claim Table:
        4       3
        2       3
        4       5
        6       7

Allocated resources:    11      7
Available resources:    -8      -3

The processes are in unsafe state.
```

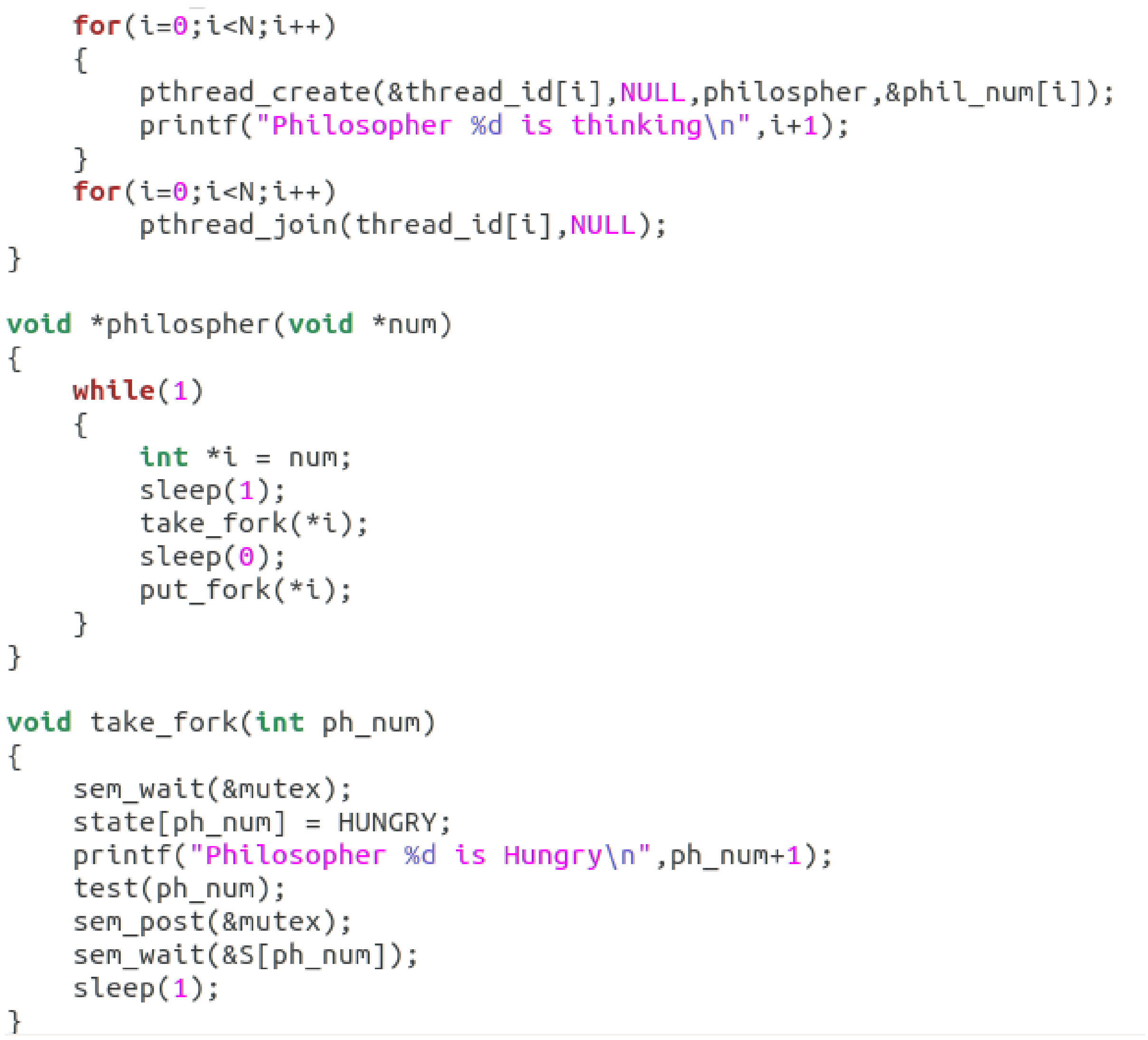
Task # 2:

Study and implement Safety algorithm and Resource-Request Algorithm in the above program.

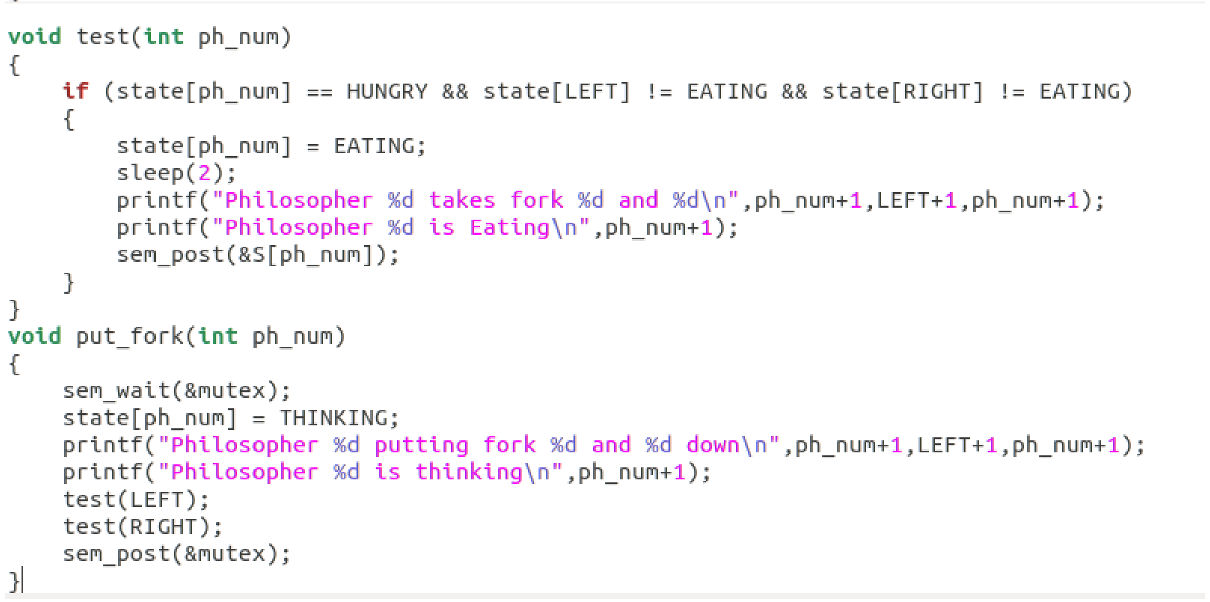
Code:



```c
#include<stdio.h>
#include<semaphore.h>
#include<pthread.h>
#include<unistd.h>

#define N 5
#define THINKING 0
#define HUNGRY 1
#define EATING 2
#define LEFT (ph_num+4)%N
#define RIGHT (ph_num+1)%N

sem_t mutex;
sem_t S[N];

void *philospher(void *num);
void take_fork(int);
void put_fork(int);
void test(int);

int state[N];
int phil_num[N]={0,1,2,3,4};

int main()
{
    int i;
    pthread_t thread_id[N];
    sem_init(&mutex,0,1);
    for(i=0;i<N;i++)
        sem_init(&S[i],0,0);
    for(i=0;i<N;i++)
    {
```

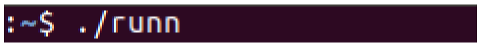


```
    for(i=0;i<N;i++)
    {
        pthread_create(&thread_id[i],NULL,philospher,&phil_num[i]);
        printf("Philosopher %d is thinking\n",i+1);
    }
    for(i=0;i<N;i++)
        pthread_join(thread_id[i],NULL);
}

void *philospher(void *num)
{
    while(1)
    {
        int *i = num;
        sleep(1);
        take_fork(*i);
        sleep(0);
        put_fork(*i);
    }
}

void take_fork(int ph_num)
{
    sem_wait(&mutex);
    state[ph_num] = HUNGRY;
    printf("Philosopher %d is Hungry\n",ph_num+1);
    test(ph_num);
    sem_post(&mutex);
    sem_wait(&S[ph_num]);
    sleep(1);
}
```

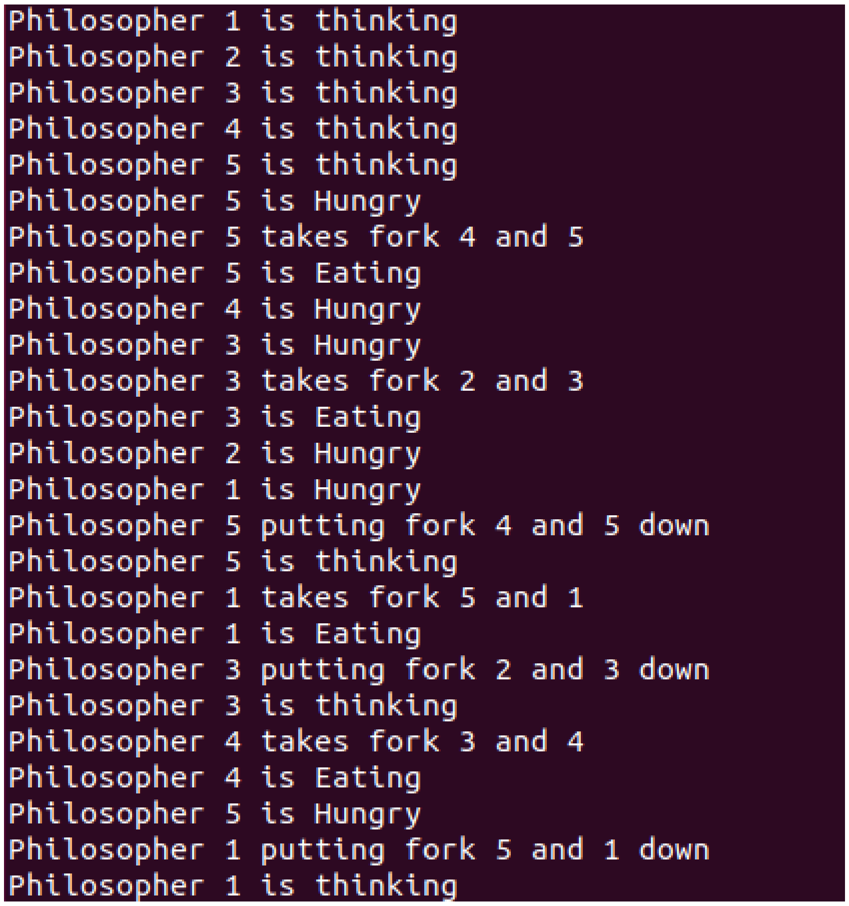


```
void test(int ph_num)
{
    if (state[ph_num] == HUNGRY && state[LEFT] != EATING && state[RIGHT] != EATING)
    {
        state[ph_num] = EATING;
        sleep(2);
        printf("Philosopher %d takes fork %d and %d\n",ph_num+1,LEFT+1,ph_num+1);
        printf("Philosopher %d is Eating\n",ph_num+1);
        sem_post(&S[ph_num]);
    }
}
void put_fork(int ph_num)
{
    sem_wait(&mutex);
    state[ph_num] = THINKING;
    printf("Philosopher %d putting fork %d and %d down\n",ph_num+1,LEFT+1,ph_num+1);
    printf("Philosopher %d is thinking\n",ph_num+1);
    test(LEFT);
    test(RIGHT);
    sem_post(&mutex);
}
```

Output:



```
:~$ ./runn
```



```
Philosopher 1 is thinking
Philosopher 2 is thinking
Philosopher 3 is thinking
Philosopher 4 is thinking
Philosopher 5 is thinking
Philosopher 5 is Hungry
Philosopher 5 takes fork 4 and 5
Philosopher 5 is Eating
Philosopher 4 is Hungry
Philosopher 3 is Hungry
Philosopher 3 takes fork 2 and 3
Philosopher 3 is Eating
Philosopher 2 is Hungry
Philosopher 1 is Hungry
Philosopher 5 putting fork 4 and 5 down
Philosopher 5 is thinking
Philosopher 1 takes fork 5 and 1
Philosopher 1 is Eating
Philosopher 3 putting fork 2 and 3 down
Philosopher 3 is thinking
Philosopher 4 takes fork 3 and 4
Philosopher 4 is Eating
Philosopher 5 is Hungry
Philosopher 1 putting fork 5 and 1 down
Philosopher 1 is thinking
```

BAHRIA UNIVERSITY

Karachi Campus

LAB EXPERIMENT NO. 11

LIST OF TASKS

TASK NO	OBJECTIVE
Task 1	Implement Segmentation Algorithm using c language.

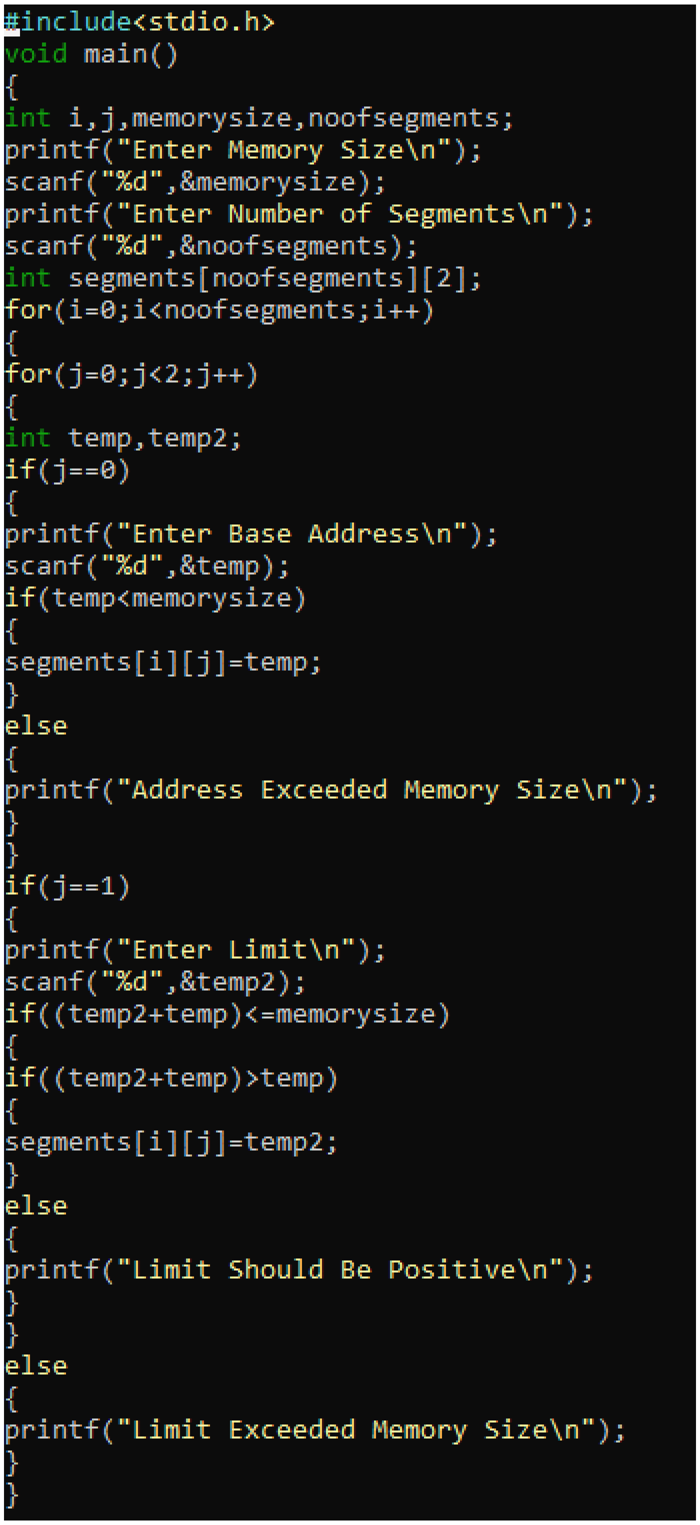
Submitted On:

30/4/2020

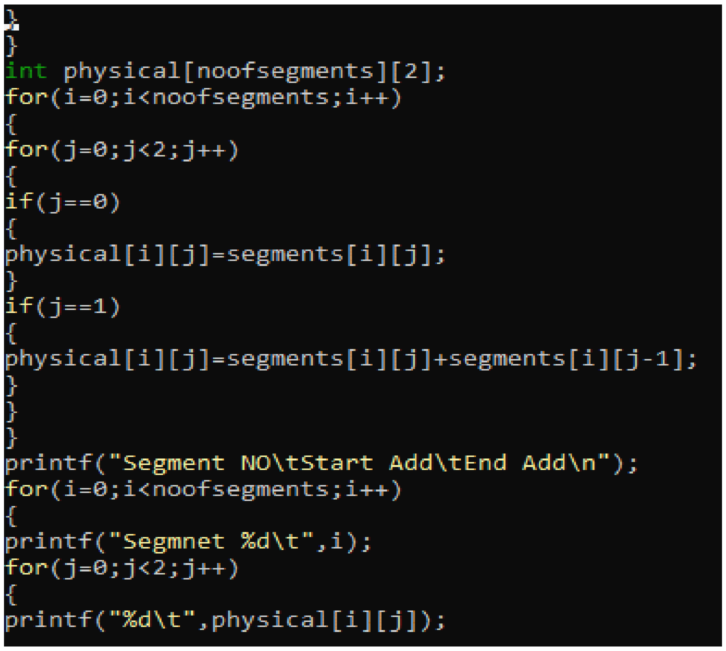
(Date: DD/MM/YY)

Task: Implement Segmentation Algorithm using c language.

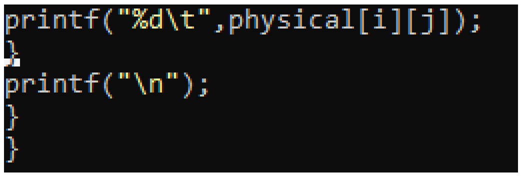
Code:



```
#include<stdio.h>
void main()
{
int i,j,memorysize,noofsegments;
printf("Enter Memory Size\n");
scanf("%d",&memorysize);
printf("Enter Number of Segments\n");
scanf("%d",&noofsegments);
int segments[noofsegments][2];
for(i=0;i<noofsegments;i++)
{
for(j=0;j<2;j++)
{
int temp,temp2;
if(j==0)
{
printf("Enter Base Address\n");
scanf("%d",&temp);
if(temp<memorysize)
{
segments[i][j]=temp;
}
else
{
printf("Address Exceeded Memory Size\n");
}
}
if(j==1)
{
printf("Enter Limit\n");
scanf("%d",&temp2);
if((temp2+temp)<=memorysize)
{
if((temp2+temp)>temp)
{
segments[i][j]=temp2;
}
else
{
printf("Limit Should Be Positive\n");
}
}
else
{
printf("Limit Exceeded Memory Size\n");
}
}
```

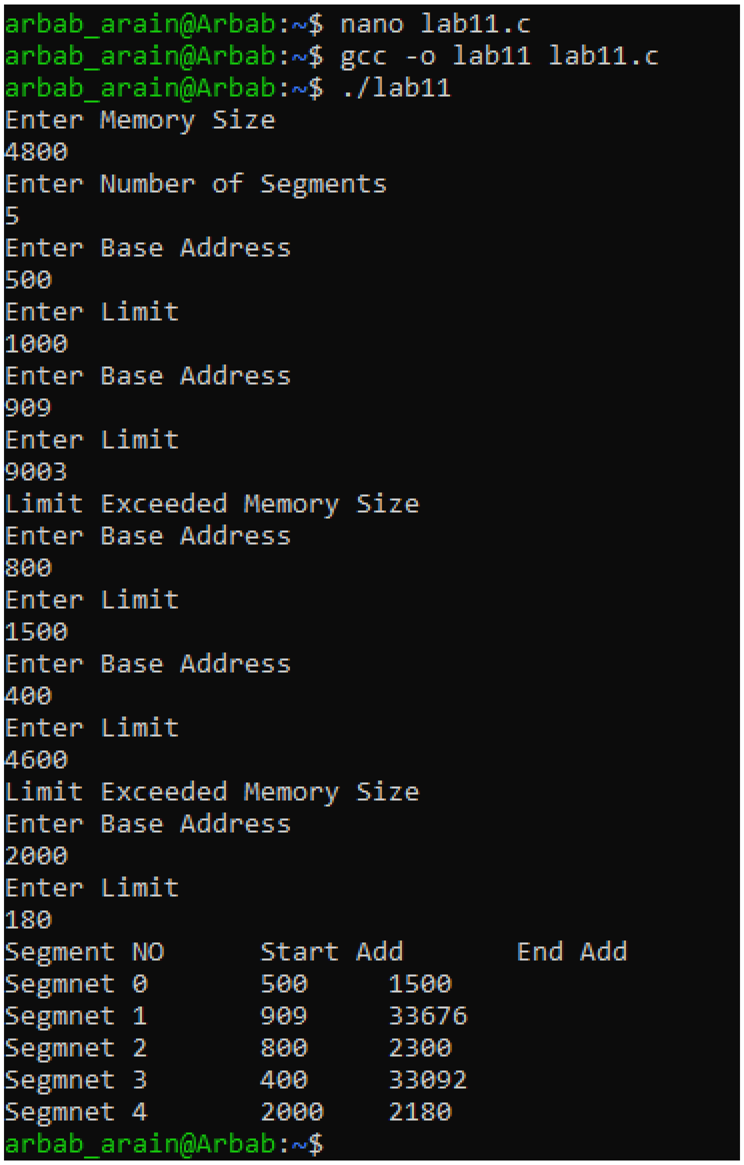


```
}
}
int physical[noofsegments][2];
for(i=0;i<noofsegments;i++)
{
for(j=0;j<2;j++)
{
if(j==0)
{
physical[i][j]=segments[i][j];
}
if(j==1)
{
physical[i][j]=segments[i][j]+segments[i][j-1];
}
}
}
printf("Segment NO\tStart Add\tEnd Add\n");
for(i=0;i<noofsegments;i++)
{
printf("Segmnet %d\t",i);
for(j=0;j<2;j++)
{
printf("%d\t",physical[i][j]);
```



```
printf("%d\t",physical[i][j]);
}
printf("\n");
}
}
```

Output:



```
arbab_arain@Arbab:~$ nano lab11.c
arbab_arain@Arbab:~$ gcc -o lab11 lab11.c
arbab_arain@Arbab:~$ ./lab11
Enter Memory Size
4800
Enter Number of Segments
5
Enter Base Address
500
Enter Limit
1000
Enter Base Address
909
Enter Limit
9003
Limit Exceeded Memory Size
Enter Base Address
800
Enter Limit
1500
Enter Base Address
400
Enter Limit
4600
Limit Exceeded Memory Size
Enter Base Address
2000
Enter Limit
180
Segment NO      Start Add       End Add
Segmnet 0       500     1500
Segmnet 1       909     33676
Segmnet 2       800     2300
Segmnet 3       400     33092
Segmnet 4       2000    2180
arbab_arain@Arbab:~$
```

BAHRIA UNIVERSITY

Karachi Campus

LAB EXPERIMENT NO. 12
LIST OF TASKS

TASK NO	OBJECTIVE
Task 1	Implement the semaphore-based solution to Dining Philosophers' Problem explained above in C language.

Submitted On:
3/6/2020
(Date: DD/MM/YY)

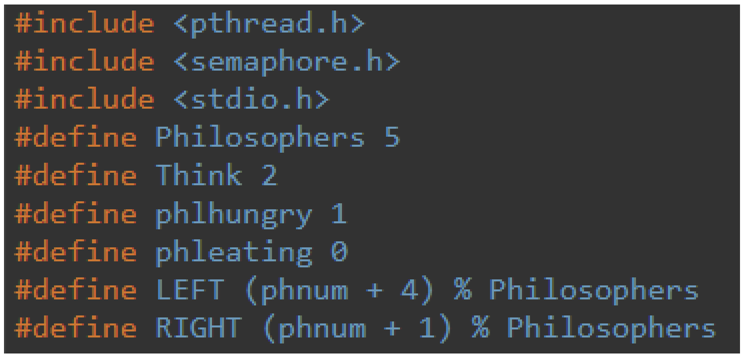
Task 1

Implement the semaphore-based solution to Dining Philosophers' Problem explained above in C language.

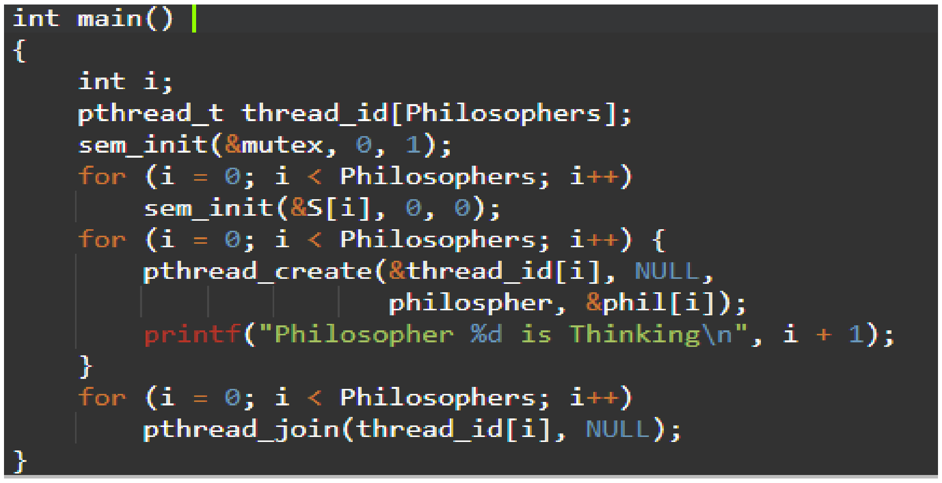
Dining Philosophers Problem:

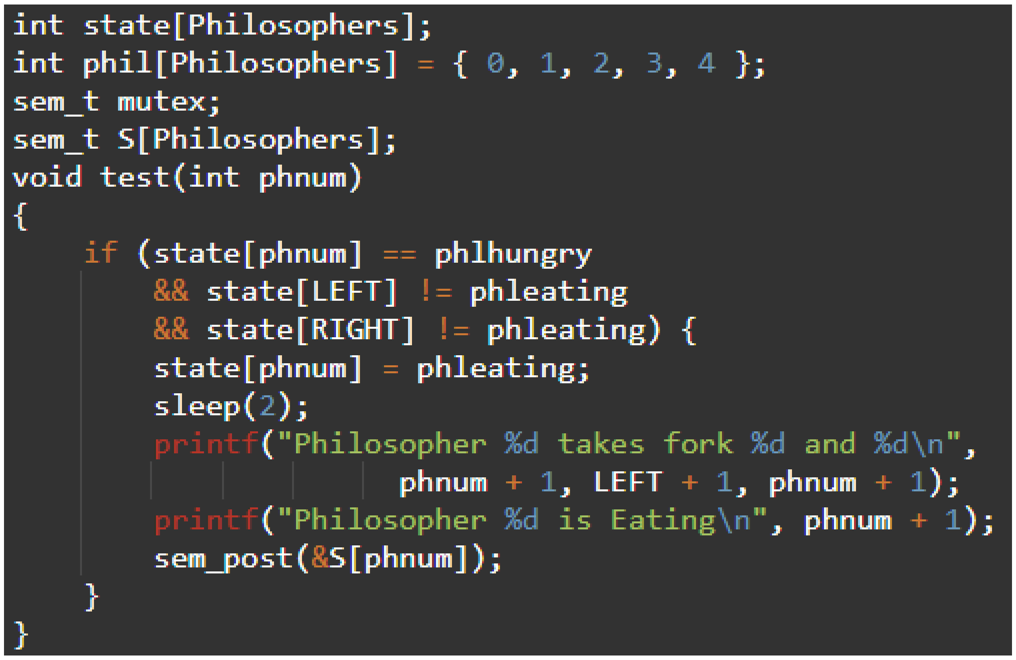
There is a dining room containing a circular table with five chairs. At each chair is a plate, and between each plate is a single chopstick. In the middle of the table is a bowl of spaghetti. Near the room are five philosophers who spend most of their time thinking, but who occasionally get hungry and need to eat so they can think some more.
A solution of the Dining Philosophers Problem is to use a semaphore to represent a chopstick. A chopstick can be picked up by executing a wait operation on the semaphore and released by executing a signal semaphore.

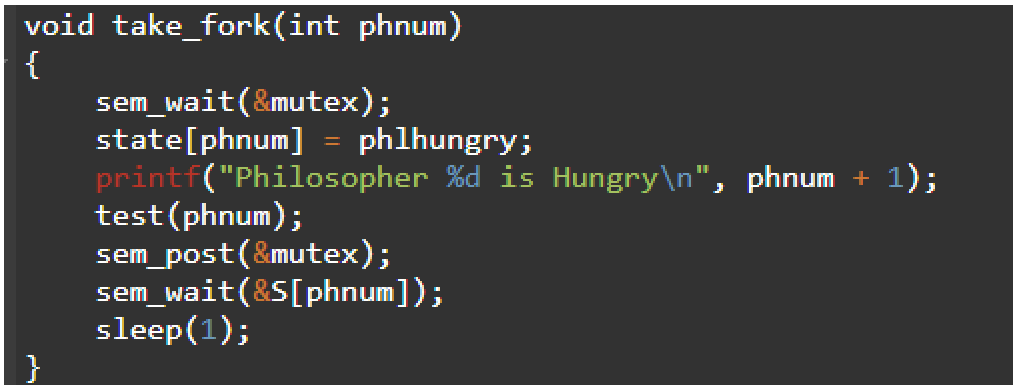
Semaphores based Implementation:

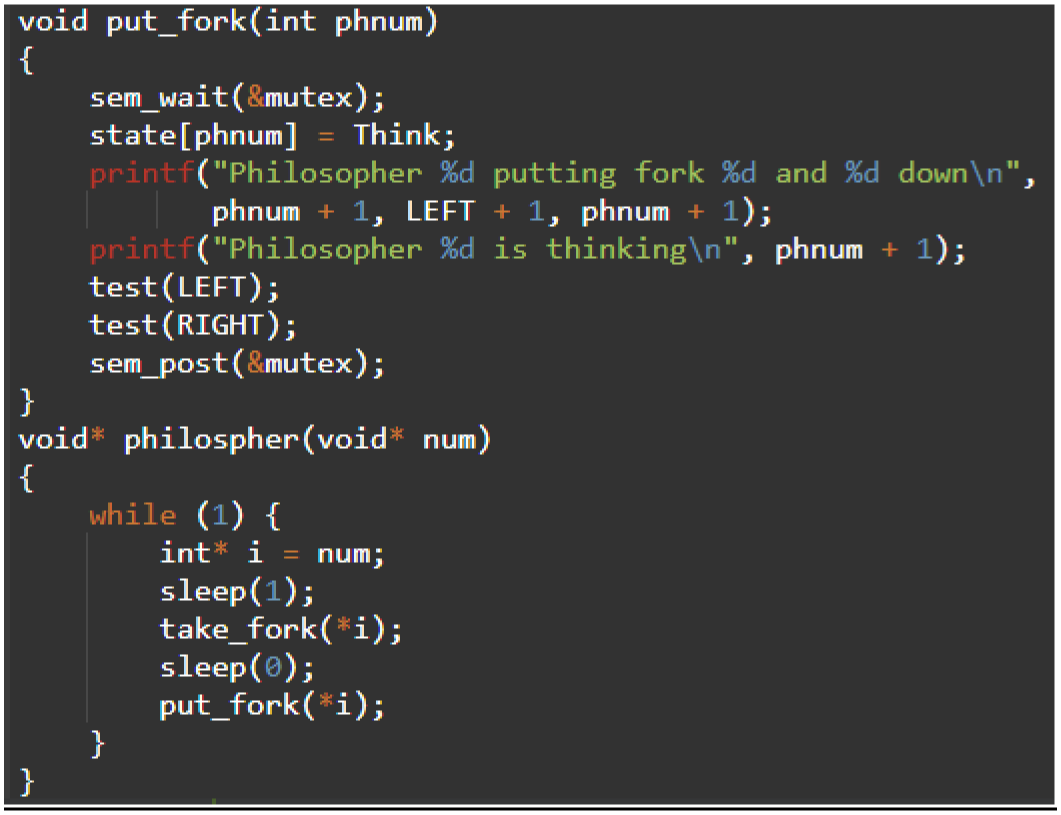


```
#include <pthread.h>
#include <semaphore.h>
#include <stdio.h>
#define Philosophers 5
#define Think 2
#define phlhungry 1
#define phleating 0
#define LEFT (phnum + 4) % Philosophers
#define RIGHT (phnum + 1) % Philosophers
```


```c
int main()
{
    int i;
    pthread_t thread_id[Philosophers];
    sem_init(&mutex, 0, 1);
    for (i = 0; i < Philosophers; i++)
        sem_init(&S[i], 0, 0);
    for (i = 0; i < Philosophers; i++) {
        pthread_create(&thread_id[i], NULL,
                       philospher, &phil[i]);
        printf("Philosopher %d is Thinking\n", i + 1);
    }
    for (i = 0; i < Philosophers; i++)
        pthread_join(thread_id[i], NULL);
}
```

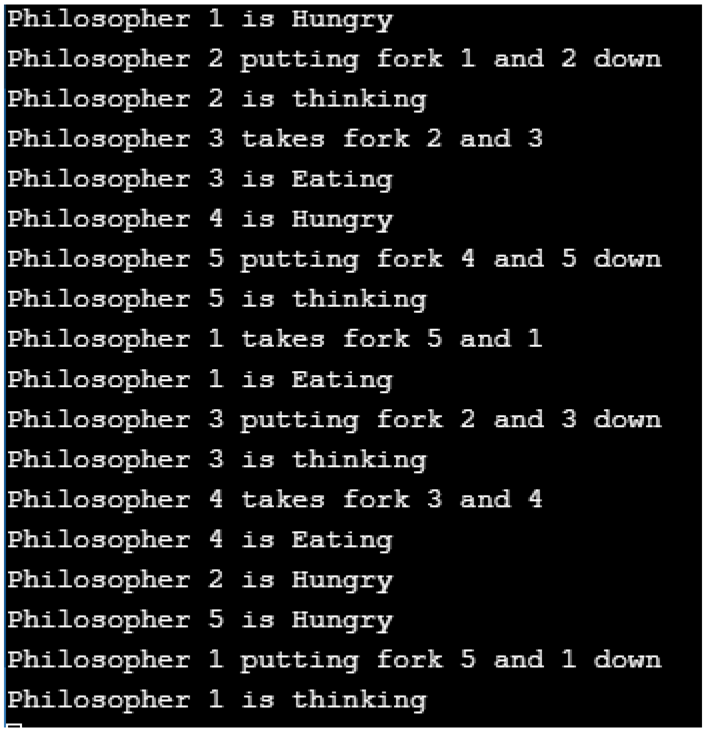

```c
int state[Philosophers];
int phil[Philosophers] = { 0, 1, 2, 3, 4 };
sem_t mutex;
sem_t S[Philosophers];
void test(int phnum)
{
    if (state[phnum] == phlhungry
        && state[LEFT] != phleating
        && state[RIGHT] != phleating) {
        state[phnum] = phleating;
        sleep(2);
        printf("Philosopher %d takes fork %d and %d\n",
                       phnum + 1, LEFT + 1, phnum + 1);
        printf("Philosopher %d is Eating\n", phnum + 1);
        sem_post(&S[phnum]);
    }
}
```


```c
void take_fork(int phnum)
{
    sem_wait(&mutex);
    state[phnum] = phlhungry;
    printf("Philosopher %d is Hungry\n", phnum + 1);
    test(phnum);
    sem_post(&mutex);
    sem_wait(&S[phnum]);
    sleep(1);
}
```



```
void put_fork(int phnum)
{
    sem_wait(&mutex);
    state[phnum] = Think;
    printf("Philosopher %d putting fork %d and %d down\n",
           phnum + 1, LEFT + 1, phnum + 1);
    printf("Philosopher %d is thinking\n", phnum + 1);
    test(LEFT);
    test(RIGHT);
    sem_post(&mutex);
}
void* philospher(void* num)
{
    while (1) {
        int* i = num;
        sleep(1);
        take_fork(*i);
        sleep(0);
        put_fork(*i);
    }
}
```

Output:



```
Philosopher 1 is Hungry
Philosopher 2 putting fork 1 and 2 down
Philosopher 2 is thinking
Philosopher 3 takes fork 2 and 3
Philosopher 3 is Eating
Philosopher 4 is Hungry
Philosopher 5 putting fork 4 and 5 down
Philosopher 5 is thinking
Philosopher 1 takes fork 5 and 1
Philosopher 1 is Eating
Philosopher 3 putting fork 2 and 3 down
Philosopher 3 is thinking
Philosopher 4 takes fork 3 and 4
Philosopher 4 is Eating
Philosopher 2 is Hungry
Philosopher 5 is Hungry
Philosopher 1 putting fork 5 and 1 down
Philosopher 1 is thinking
```

BAHRIA UNIVERSITY

Karachi Campus

LAB EXPERIMENT NO. 13

LIST OF TASKS

TASK NO	OBJECTIVE
	Following the code guideline of Best-Fit Placement algorithm, write the C language program for First-Fit and Next-Fit Placement algorithms.

Submitted On:

3/6/2020

(Date: DD/MM/YY)

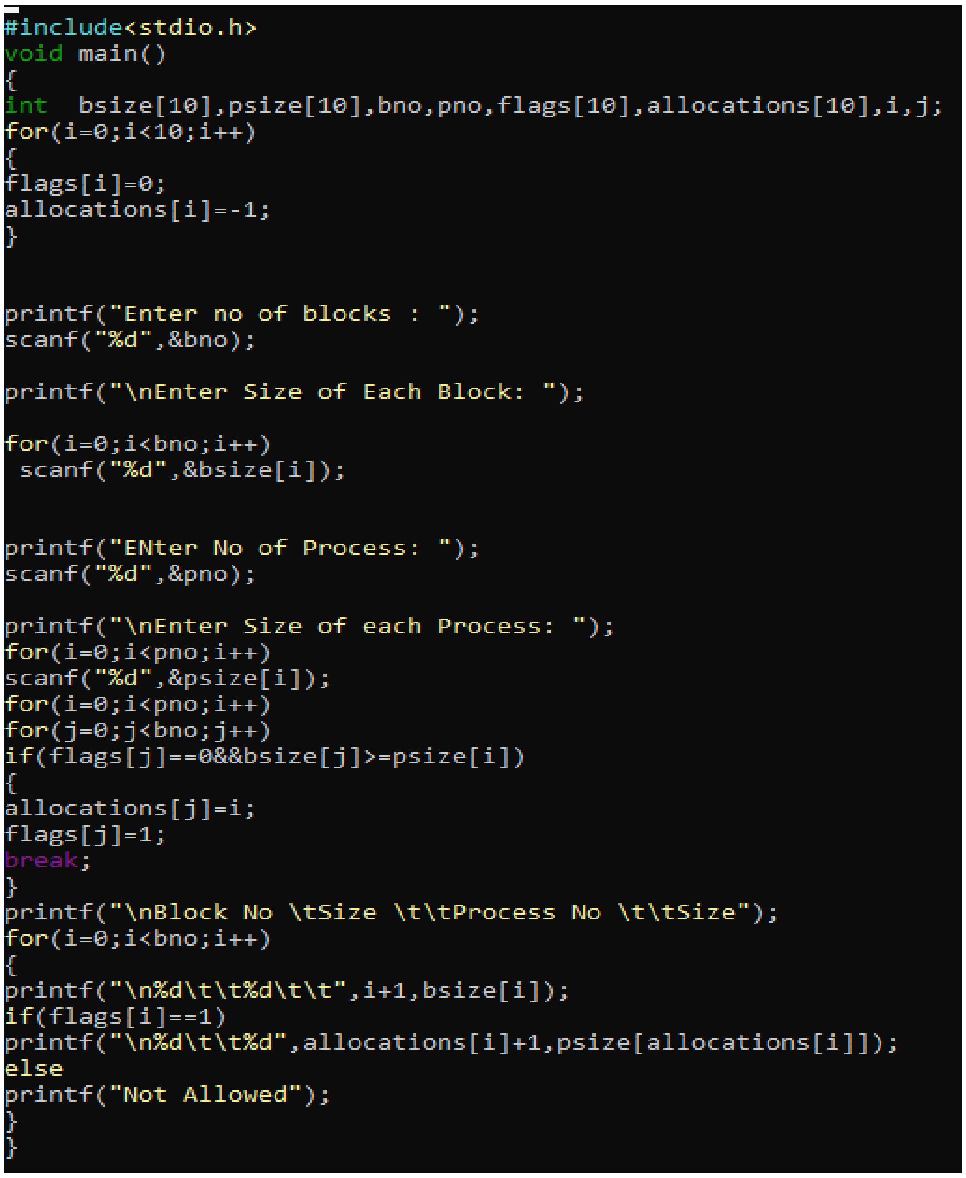
Task:

Following the code guideline of Best-Fit Placement algorithm, write the C language program for First-Fit and Next-Fit Placement algorithms.

First-Fit Algorithm:

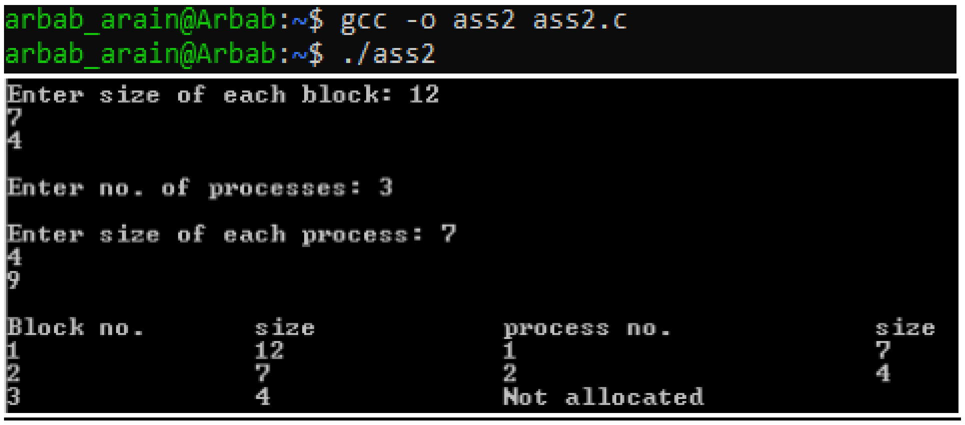
In this scheme we check the blocks in a sequential manner which means we pick the first process then compare it's size with first block size if it is less than size of block it is allocated otherwise we move to second block and so on.

Implementation:



```
#include<stdio.h>
void main()
{
int  bsize[10],psize[10],bno,pno,flags[10],allocations[10],i,j;
for(i=0;i<10;i++)
{
flags[i]=0;
allocations[i]=-1;
}


printf("Enter no of blocks : ");
scanf("%d",&bno);

printf("\nEnter Size of Each Block: ");

for(i=0;i<bno;i++)
 scanf("%d",&bsize[i]);


printf("ENter No of Process: ");
scanf("%d",&pno);

printf("\nEnter Size of each Process: ");
for(i=0;i<pno;i++)
scanf("%d",&psize[i]);
for(i=0;i<pno;i++)
for(j=0;j<bno;j++)
if(flags[j]==0&&bsize[j]>=psize[i])
{
allocations[j]=i;
flags[j]=1;
break;
}
printf("\nBlock No \tSize \t\tProcess No \t\tSize");
for(i=0;i<bno;i++)
{
printf("\n%d\t\t%d\t\t",i+1,bsize[i]);
if(flags[i]==1)
printf("\n%d\t\t%d",allocations[i]+1,psize[allocations[i]]);
else
printf("Not Allowed");
}
}
```

Output:



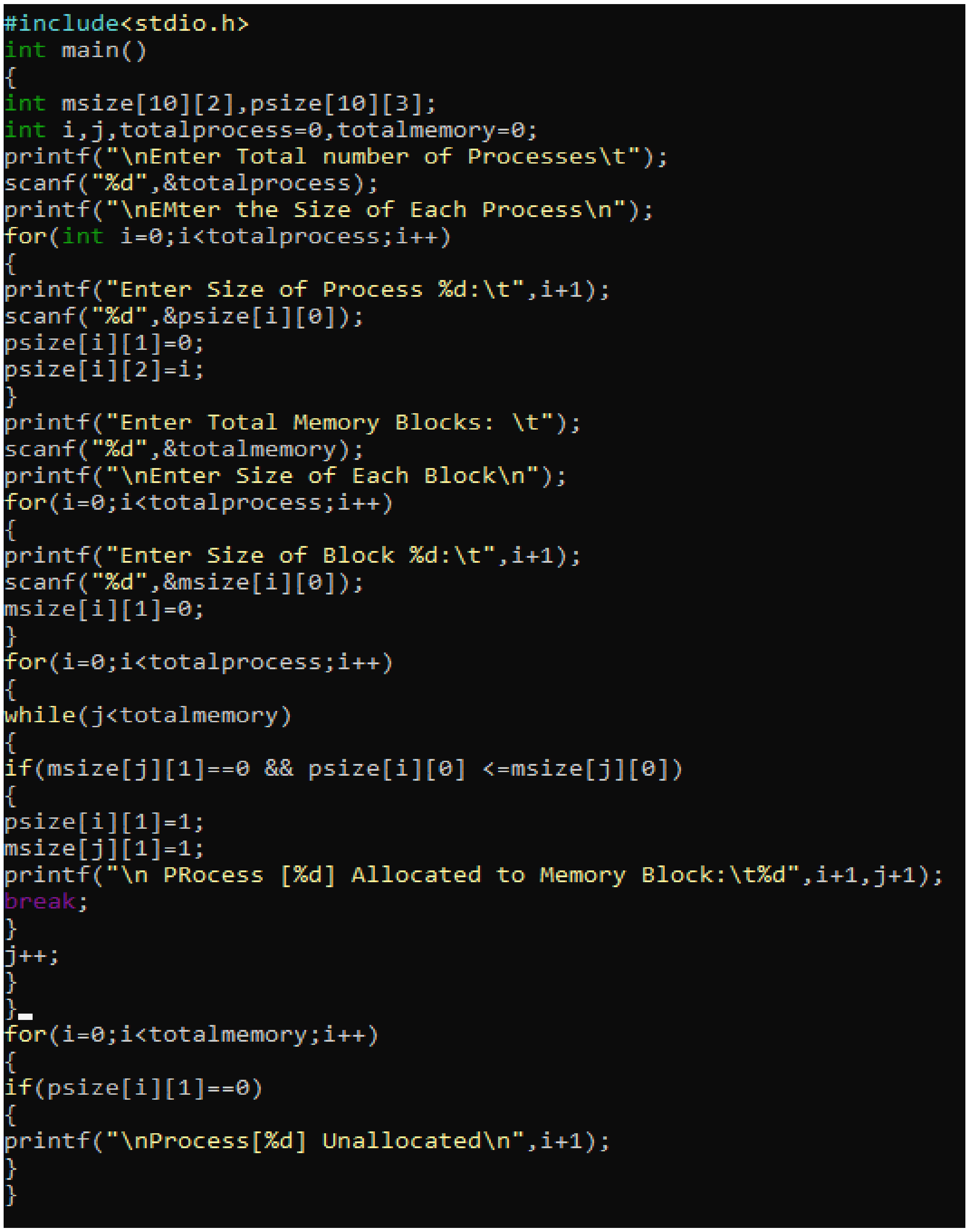
```
arbab_arain@Arbab:~$ gcc -o ass2 ass2.c
arbab_arain@Arbab:~$ ./ass2
```

```
Enter size of each block: 12
7
4

Enter no. of processes: 3

Enter size of each process: 7
4
9

Block no.       size            process no.             size
1               12              1                       7
2               7               2                       4
3               4               Not allocated
```

Next-Fit Algorithm:

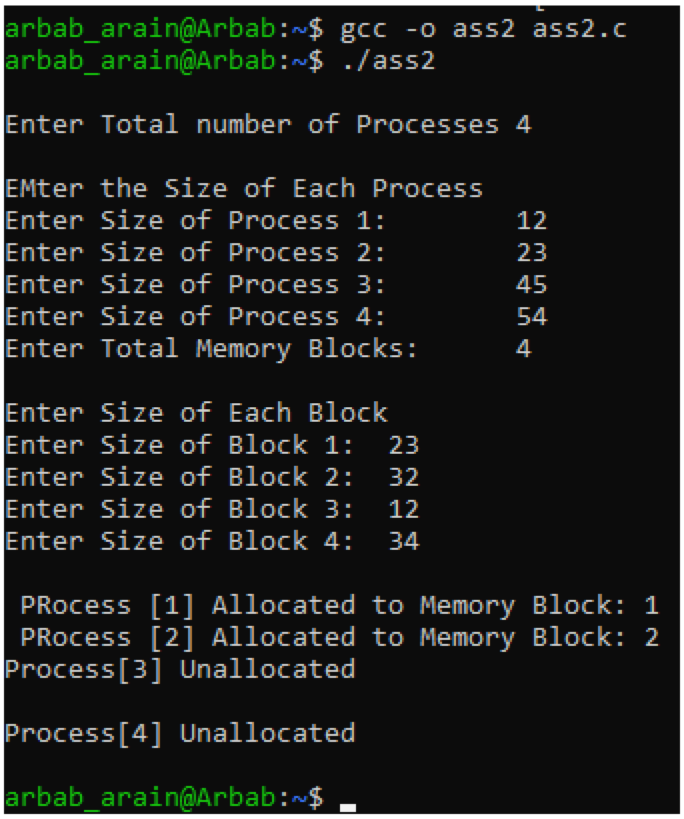
This algorithm keeps a track of the positions where every file is written in the memory. It then allocates the very next available memory block to the succeeding processes. So, when a process is executed to be stored in the memory, the previous bin or the memory block is checked for its availability.

Implementation:



```
#include<stdio.h>
int main()
{
int msize[10][2],psize[10][3];
int i,j,totalprocess=0,totalmemory=0;
printf("\nEnter Total number of Processes\t");
scanf("%d",&totalprocess);
printf("\nEMter the Size of Each Process\n");
for(int i=0;i<totalprocess;i++)
{
printf("Enter Size of Process %d:\t",i+1);
scanf("%d",&psize[i][0]);
psize[i][1]=0;
psize[i][2]=i;
}
printf("Enter Total Memory Blocks: \t");
scanf("%d",&totalmemory);
printf("\nEnter Size of Each Block\n");
for(i=0;i<totalprocess;i++)
{
printf("Enter Size of Block %d:\t",i+1);
scanf("%d",&msize[i][0]);
msize[i][1]=0;
}
for(i=0;i<totalprocess;i++)
{
while(j<totalmemory)
{
if(msize[j][1]==0 && psize[i][0] <=msize[j][0])
{
psize[i][1]=1;
msize[j][1]=1;
printf("\n PRocess [%d] Allocated to Memory Block:\t%d",i+1,j+1);
break;
}
j++;
}
}
for(i=0;i<totalmemory;i++)
{
if(psize[i][1]==0)
{
printf("\nProcess[%d] Unallocated\n",i+1);
}
}
```

Output:



```
arbab_arain@Arbab:~$ gcc -o ass2 ass2.c
arbab_arain@Arbab:~$ ./ass2

Enter Total number of Processes 4

EMter the Size of Each Process
Enter Size of Process 1:        12
Enter Size of Process 2:        23
Enter Size of Process 3:        45
Enter Size of Process 4:        54
Enter Total Memory Blocks:      4

Enter Size of Each Block
Enter Size of Block 1:  23
Enter Size of Block 2:  32
Enter Size of Block 3:  12
Enter Size of Block 4:  34

 PRocess [1] Allocated to Memory Block: 1
 PRocess [2] Allocated to Memory Block: 2
Process[3] Unallocated

Process[4] Unallocated

arbab_arain@Arbab:~$
```