Name : Vaibhav Gorakh Lonkar

PRN : 21410027

Batch : EN2

RTOS Lab Experiment No. 5

Title: Use of Makefile.

Commands used :

1.touch *filename*

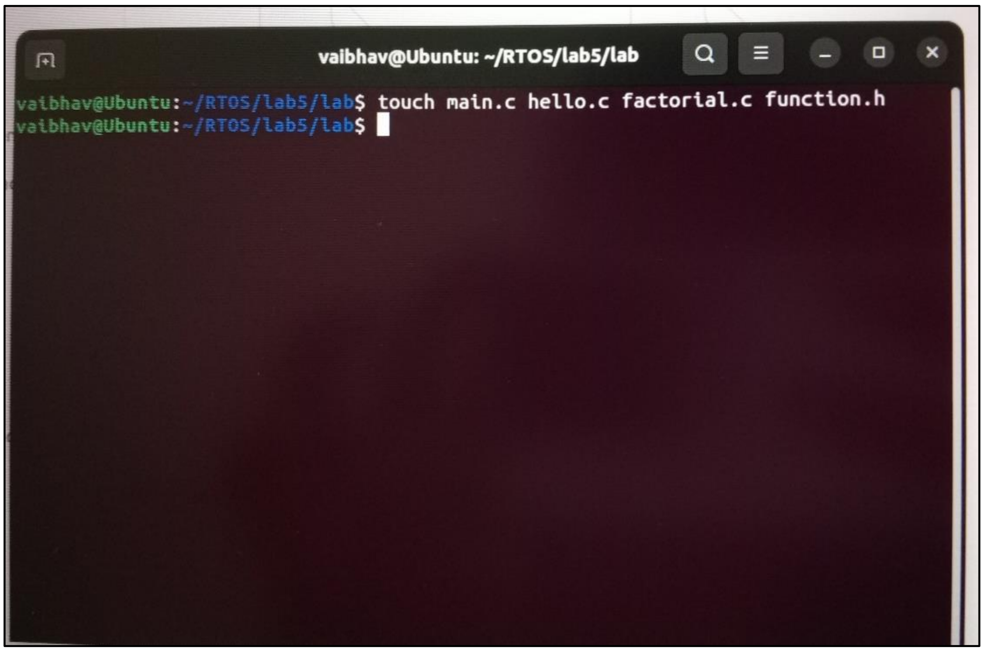
2.gcc *filename* -o *outputfilename*

3.touch *Makefile*

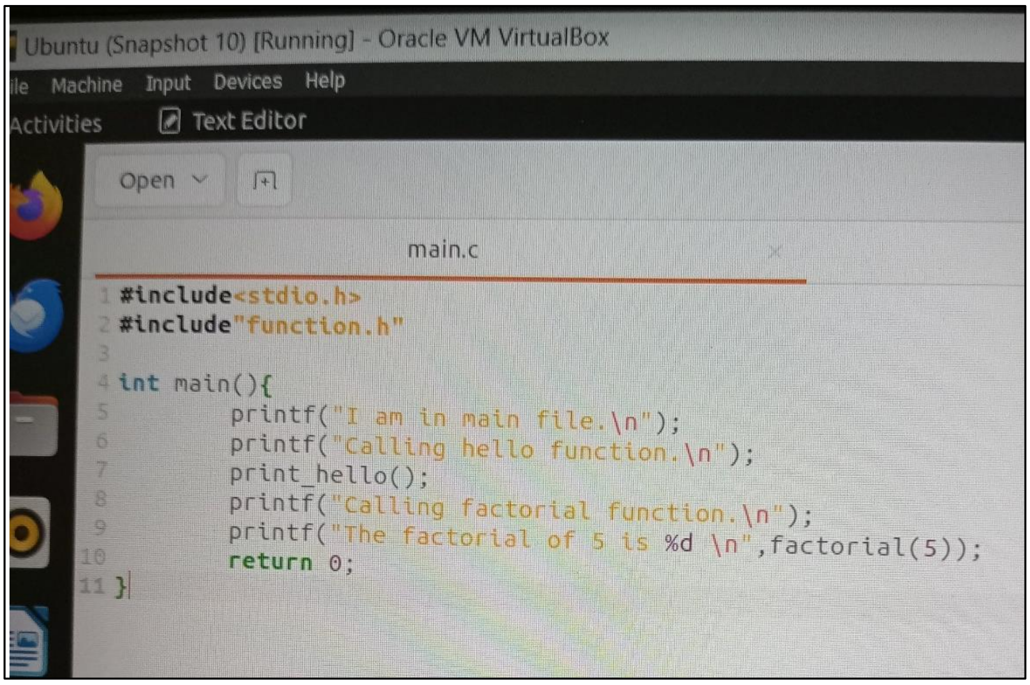
4.make

5../*outputfilename*

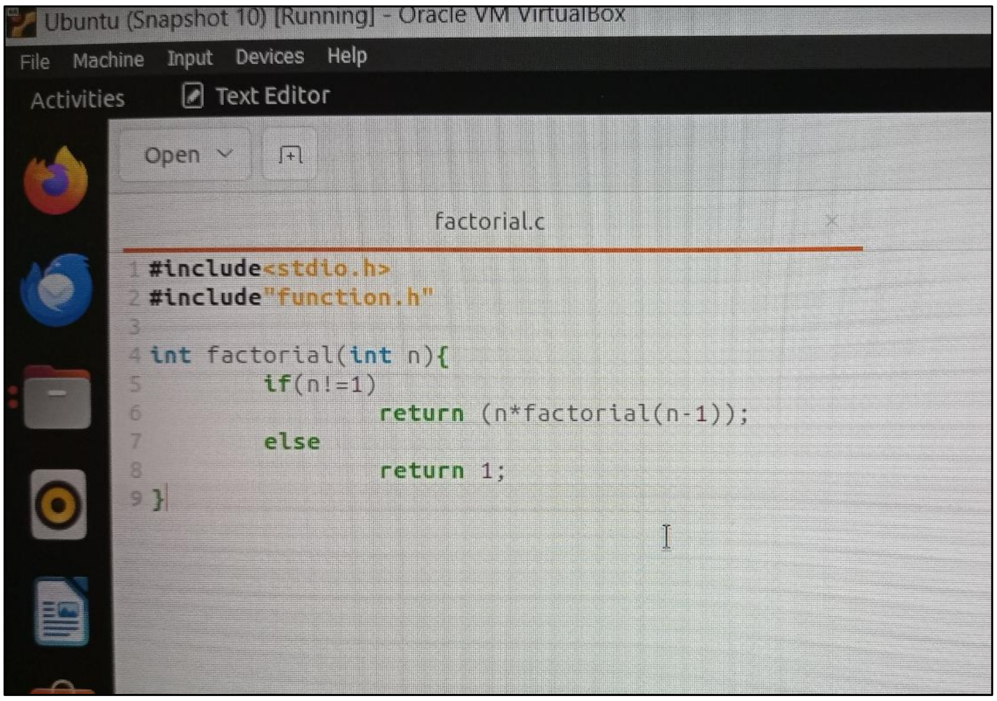
Procedure:



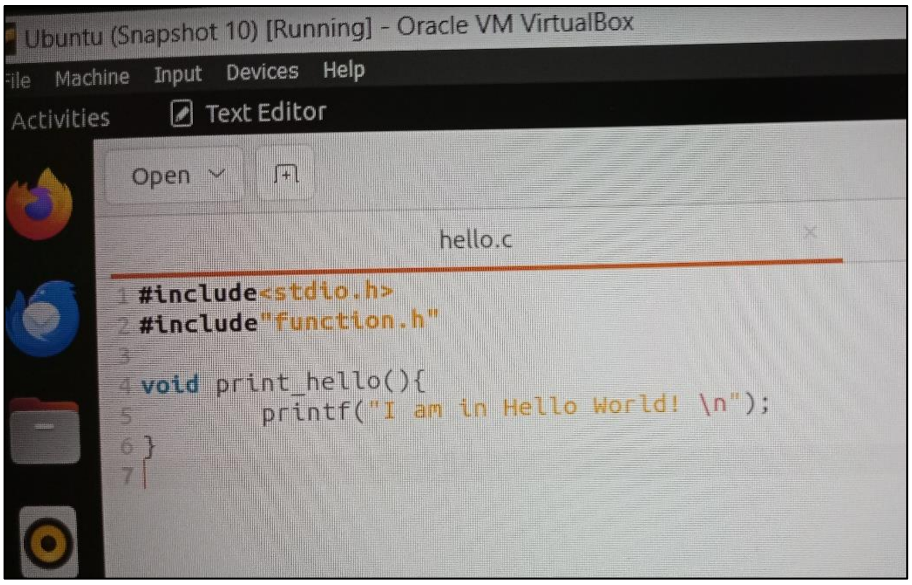
```
vaibhav@Ubuntu:~/RTOS/lab5/lab$ touch main.c hello.c factorial.c function.h
vaibhav@Ubuntu:~/RTOS/lab5/lab$
```


Ubuntu (Snapshot 10) [Running] - Oracle VM VirtualBox
ile Machine Input Devices Help
Activities
Text Editor
Open
main.c

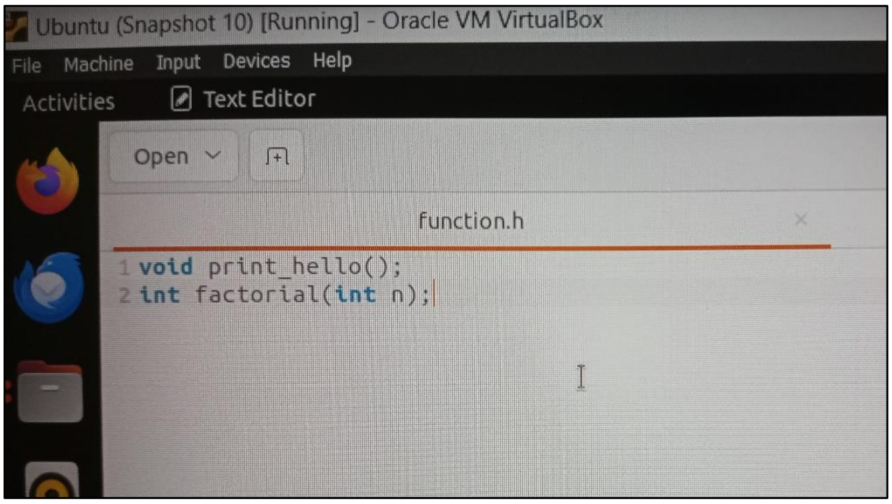
```c
#include<stdio.h>
#include"function.h"

int main(){
	printf("I am in main file.\n");
	printf("Calling hello function.\n");
	print_hello();
	printf("Calling factorial function.\n");
	printf("The factorial of 5 is %d \n",factorial(5));
	return 0;
}
```


Ubuntu (Snapshot 10) [Running] - Oracle VM VirtualBox
File Machine Input Devices Help
Activities
Text Editor
Open
factorial.c

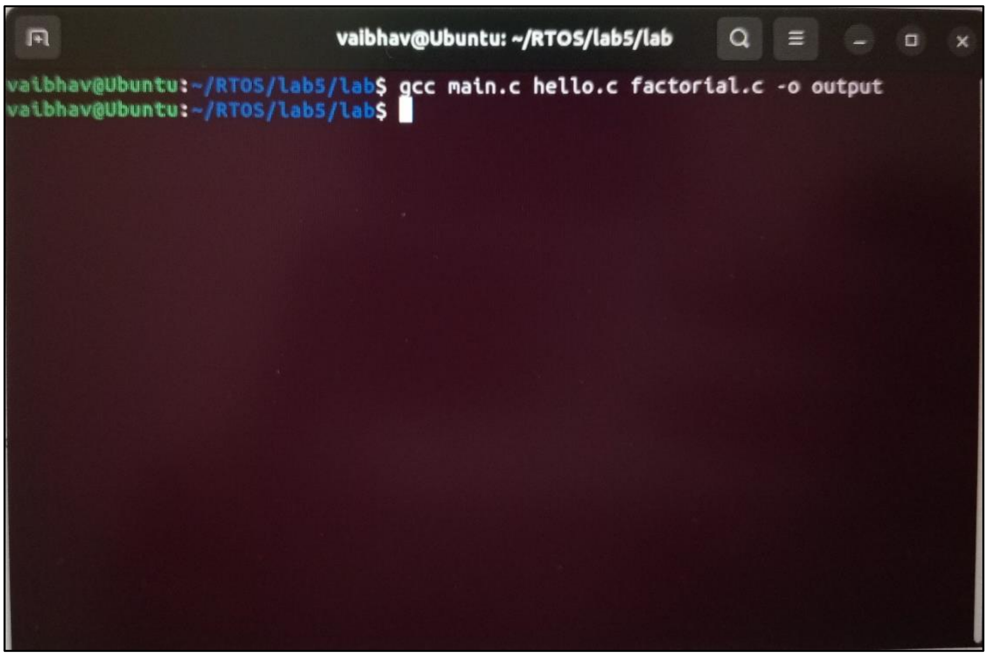
```c
#include<stdio.h>
#include"function.h"

int factorial(int n){
	if(n!=1)
		return (n*factorial(n-1));
	else
		return 1;
}
```


Ubuntu (Snapshot 10) [Running] - Oracle VM VirtualBox
File Machine Input Devices Help
Activities Text Editor
Open
hello.c

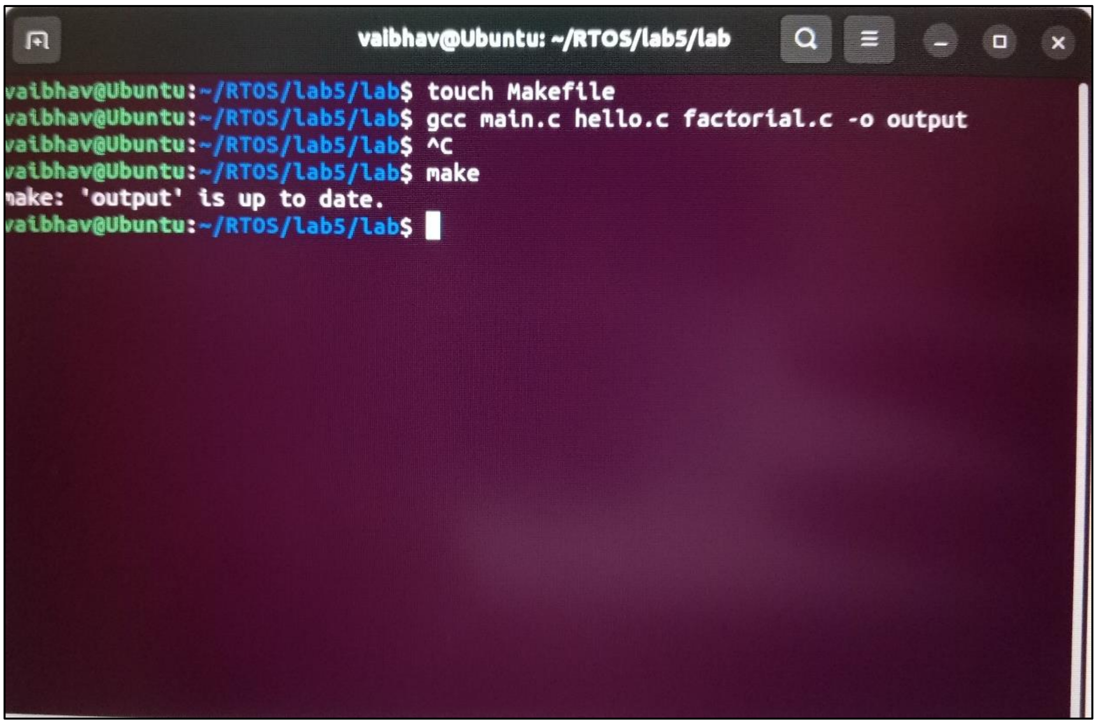
```c
#include<stdio.h>
#include"function.h"

void print_hello(){
        printf("I am in Hello World! \n");
}

```


Ubuntu (Snapshot 10) [Running] - Oracle VM VirtualBox
File Machine Input Devices Help
Activities Text Editor
Open
function.h

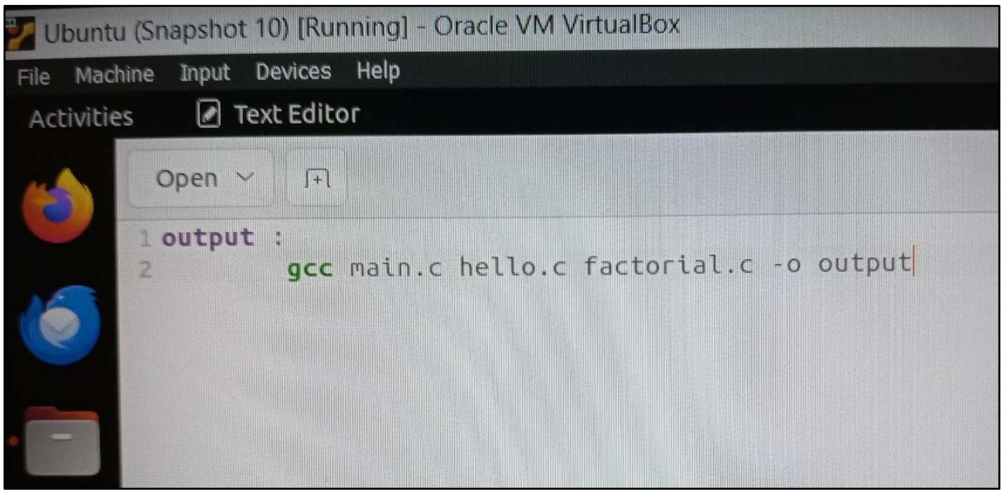
```c
void print_hello();
int factorial(int n);
```



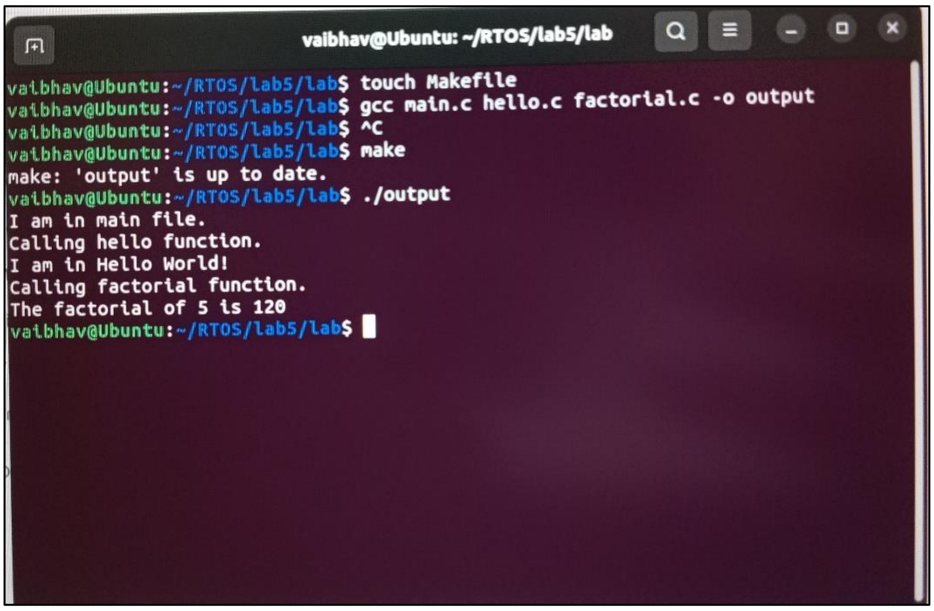
```
vaibhav@Ubuntu:~/RTOS/lab5/lab$ gcc main.c hello.c factorial.c -o output
vaibhav@Ubuntu:~/RTOS/lab5/lab$ 
```



```
vaibhav@Ubuntu:~/RTOS/lab5/lab$ touch Makefile
vaibhav@Ubuntu:~/RTOS/lab5/lab$ gcc main.c hello.c factorial.c -o output
vaibhav@Ubuntu:~/RTOS/lab5/lab$ ^C
vaibhav@Ubuntu:~/RTOS/lab5/lab$ make
make: 'output' is up to date.
vaibhav@Ubuntu:~/RTOS/lab5/lab$
```


Ubuntu (Snapshot 10) [Running] - Oracle VM VirtualBox
File Machine Input Devices Help
Activities
Text Editor
Open

```
output :
        gcc main.c hello.c factorial.c -o output
```



```
vaibhav@Ubuntu:~/RTOS/lab5/lab$ touch Makefile
vaibhav@Ubuntu:~/RTOS/lab5/lab$ gcc main.c hello.c factorial.c -o output
vaibhav@Ubuntu:~/RTOS/lab5/lab$ ^C
vaibhav@Ubuntu:~/RTOS/lab5/lab$ make
make: 'output' is up to date.
vaibhav@Ubuntu:~/RTOS/lab5/lab$ ./output
I am in main file.
Calling hello function.
I am in Hello World!
Calling factorial function.
The factorial of 5 is 120
vaibhav@Ubuntu:~/RTOS/lab5/lab$
```

Conclusion :

In conclusion, Makefiles are an essential tool for automating and managing the build process in software development. By defining clear rules for compiling and linking code, they streamline workflows, enhance efficiency, and ensure consistency across different environments. Mastering Makefiles not only simplifies complex builds but also contributes to better project organization and reduced risk of errors. Embracing Makefiles can significantly improve productivity and maintainability in software projects.

Answer the following:

Q.Significance of Makefile

→

The **significance of a Makefile** in software development, especially in Linux environments, is substantial for several reasons:

1. **Automation of Build Process**: A Makefile automates the process of compiling and linking code. Instead of manually typing commands every time you make changes, you can just run make to compile the program.
2. **Dependency Management**: Makefiles automatically handle dependencies between files. When you change a source file, make only recompiles the parts of the project affected by that change, which saves time.
3. **Platform Independence**: Makefiles allow developers to write cross-platform build scripts. They can define different compilation rules for different environments, making it easier to compile the project on various systems.
4. **Efficient Project Management**: Makefiles make it easier to manage large projects with many source files. They provide a structured way to organize the build process and keep track of which files depend on others.
5. **Customization**: Developers can define custom rules for running tests, cleaning up files, or deploying software, making it a flexible tool beyond just compilation.
6. **Reproducibility**: By using a Makefile, the build process becomes consistent and repeatable across different machines and environments, reducing the risk of errors due to manual processes.