

Name : Vaibhav Gorakh Lonkar

Batch : EN2

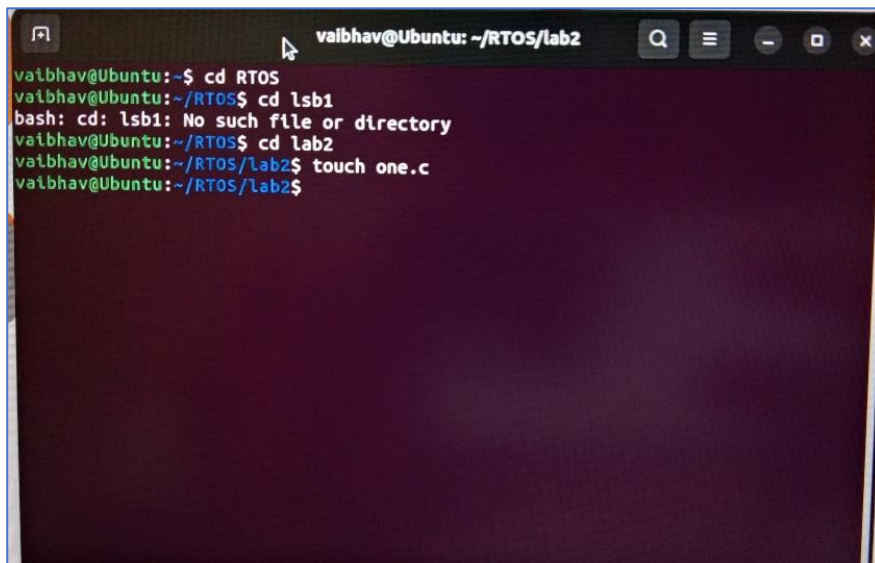
PRN : 21410027

RTOS Lab Experiment No. 2

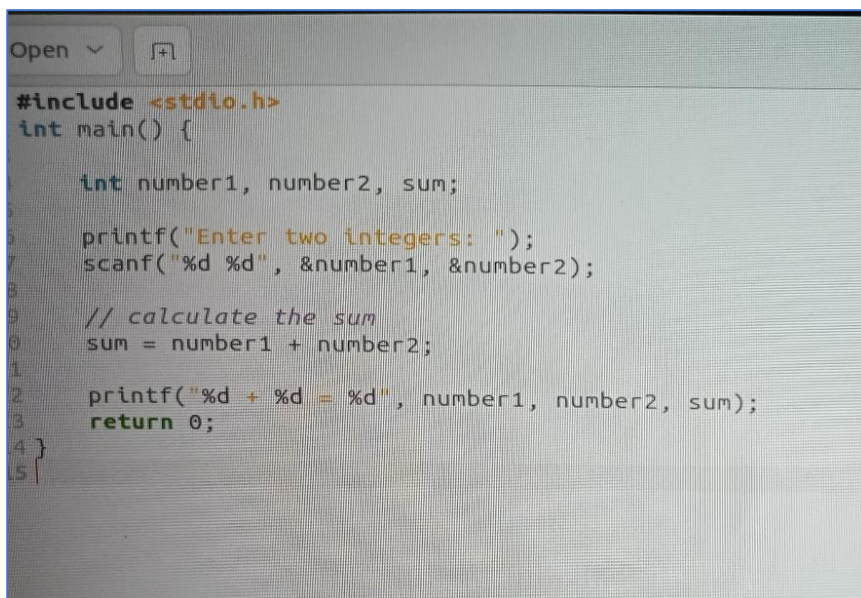
Title: Execution of C program in Linux.

Part A: Single file C program.

→



```
vaibhav@Ubuntu: ~/RTOS/lab2
vaibhav@Ubuntu:~$ cd RTOS
vaibhav@Ubuntu:~/RTOS$ cd lsb1
bash: cd: lsb1: No such file or directory
vaibhav@Ubuntu:~/RTOS$ cd lab2
vaibhav@Ubuntu:~/RTOS/lab2$ touch one.c
vaibhav@Ubuntu:~/RTOS/lab2$
```



```
Open ▾
#include <stdio.h>
int main() {

    int number1, number2, sum;

    printf("Enter two integers: ");
    scanf("%d %d", &number1, &number2);

    // calculate the sum
    sum = number1 + number2;

    printf("%d + %d = %d", number1, number2, sum);
    return 0;
}
```

```
vaibhav@Ubuntu: ~/RTOS/lab2
vaibhav@Ubuntu:~/RTOS/lab2$ gcc one.c
vaibhav@Ubuntu:~/RTOS/lab2$ ./one
bash: ./one: No such file or directory
vaibhav@Ubuntu:~/RTOS/lab2$ gcc one.c -o
gcc: error: missing filename after '-o'
vaibhav@Ubuntu:~/RTOS/lab2$ gcc one.c -o one
vaibhav@Ubuntu:~/RTOS/lab2$ ./one
Enter two integers: 12
34
12 + 34 = 46vaibhav@Ubuntu:~/RTOS/lab2$
```


Part B: Multifile C program

→

```
vaibhav@Ubuntu: ~/RTOS/lab2
vaibhav@Ubuntu:~/RTOS/lab2$ touch two.c
vaibhav@Ubuntu:~/RTOS/lab2$ touch three.c
vaibhav@Ubuntu:~/RTOS/lab2$ touch four.c
vaibhav@Ubuntu:~/RTOS/lab2$
```








```
1 #include <stdio.h>
2 int main() {
3
4     int number1, number2, Division;
5     Division = number1 / number2;
6
7     printf("%d / %d = %d", number1, number2, Division);
8     return 0;
9 }
10 |
```

Open ▾ 

```
1 #include <stdio.h>
2 int main() {
3
4     int number1=30, number2=10, sub;
5
6     sub = number1 - number2;
7
8     printf("%d - %d = %d", number1, number2, sub);
9     return 0;
10 }
11 |
```



```
Open ▾ 
1 #include <stdio.h>
2 int main() {
3
4     int number1, number2, multiplication;
5     multiplication = number1 * number2;
6
7     printf("%d * %d = %d", number1, number2, multiplication);
8     return 0;
9 }
10 |
```

```
vaibhav@Ubuntu: ~/RTOS/lab2    
vaibhav@Ubuntu:~/RTOS/lab2$ gcc four.c -o four
vaibhav@Ubuntu:~/RTOS/lab2$ gcc one.c -o one
vaibhav@Ubuntu:~/RTOS/lab2$ gcc two.c -o two
vaibhav@Ubuntu:~/RTOS/lab2$ gcc three.c -o three
vaibhav@Ubuntu:~/RTOS/lab2$ ./one && ./two && ./three && ./four
0 + 10 = 40 /n30 - 10 = 20 /n30 * 10 = 300 /n30 / 10 = 3 /nvaibhav@Ubuntu:
RTOS/lab2$
```

Answer the following:

Write meaning and use of GCC in Linux.

→

GCC stands for the **GNU Compiler Collection**. It's a powerful compiler system produced by the GNU Project, primarily used for compiling C, C++, and other programming languages on Linux and other Unix-like operating systems. GCC is the default compiler on most Linux distributions and is crucial for building software from source code.

Use of GCC in Linux:

- **Compiling Programs:** GCC compiles source code written in languages like C or C++ into executable binaries.
- **Cross-Compiling:** GCC can be used to generate code for a different platform than the one it runs on.
- **Optimization:** GCC provides various levels of optimization to improve the performance and size of the compiled code.
- **Linking:** It links together multiple object files and libraries into a single executable.
- **Debugging Support:** GCC supports generating debugging information that can be used with debugging tools like GDB.

Write advantages and disadvantages of multifile C programming.

→ **Advantages:**

1. **Modularity:** Breaking a program into multiple files promotes modularity. Each file can handle a specific functionality, making the code easier to manage and understand.
2. **Reusability:** Code in one file can be reused in other programs or parts of the same program, reducing redundancy.
3. **Team Collaboration:** Multiple developers can work on different files simultaneously without causing conflicts, which is useful in large projects.
4. **Faster Compilation:** When only a part of the code changes, only the affected files need to be recompiled, speeding up the build process.

Disadvantages:

1. **Complexity:** Managing multiple files can become complex, especially if there are many interdependencies between them.
2. **Linker Errors:** Multifile programs can lead to linker errors if functions or variables are not properly declared or if the wrong files are linked.
3. **Makefile Dependency:** For large projects, managing the compilation of multiple files often requires the use of a makefile, which adds an extra layer of complexity.
4. **Increased Setup Time:** Setting up a project with multiple files involves more initial setup, including organizing files and creating the necessary makefiles or build scripts.