Name: Nivedita Londhe
PRN: 22420003
Batch: EN-4
Practical No. 2

Title: Execution of C program in Linux

PART A:

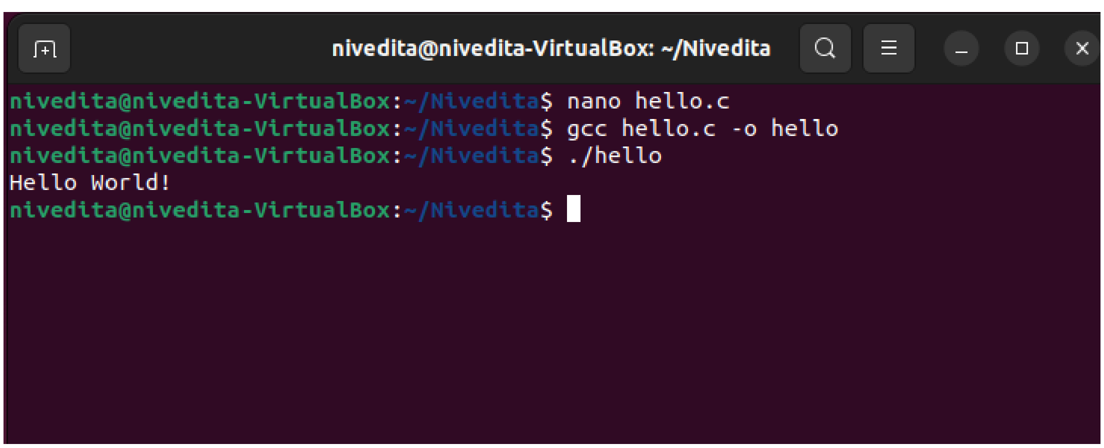
Single file Execution

- **Step 1: Created a new C file**
  I used the nano text editor to create a file named hello.c
- **Step 2: Wrote the C code**
  I entered the following code into the file and saved that.

```
#include <stdio.h>

void main() {
    printf("Hello World!\n");
}
```

- **Step 3: Compiled the code**
  I opened the a terminal, navigated to the directory containing hello.c, and used the gcc compiler with the command
- **Step 4: Executed the program**
  I ran the executable file using the following command



PART B:

Single file Execution

- **Step 1: Create Source Files**
  1. Create separate .c files for each arithmetic operation: add.c, sub.c, mul.c, and div.c.

```
nivedita@nivedita-VirtualBox: ~/Nivedita
nivedita@nivedita-VirtualBox:~/Nivedita$ touch add.c
nivedita@nivedita-VirtualBox:~/Nivedita$ touch sub.c
nivedita@nivedita-VirtualBox:~/Nivedita$ touch mul.c
nivedita@nivedita-VirtualBox:~/Nivedita$ touch div.c
nivedita@nivedita-VirtualBox:~/Nivedita$ touch main.c
nivedita@nivedita-VirtualBox:~/Nivedita$ touch arithmaticOperations.h
nivedita@nivedita-VirtualBox:~/Nivedita$
```

  2. Write the corresponding functions in each file

add.c

```c
#include <stdio.h>

int add(int a, int b) {
    return a + b;
}
```

sub.c

```c
#include <stdio.h>

int sub(int a, int b) {
    return a - b;
}
```

mul.c

```c
#include <stdio.h>

int multiply(int a, int b) {
    return a * b;
}
```

div.c

```
#include <stdio.h>

int divide(int a, int b) {
    if (b == 0) {
        printf("Error: Division by zero!\n");
        return 0;
    }
    return a / b;
}
```

- **Step 2: Create Header File**
  1. Create header file.
  2. Declare the functions in this header file.

  arithmaticOperations.h

```
int add(int a, int b);
int subtract(int a, int b);
int multiply(int a, int b);
int divide(int a, int b);
```

- **Step 3: Write the Main Program**
  1. Create a main.c file to handle user input, call the arithmetic functions, and display the results.
  2. Include the necessary header files in main.c.

  Main.c

```
#include <stdio.h>
#include "arithmaticOperations.h"

int main() {
    int num1, num2;

    printf("Enter two numbers: ");
    scanf("%d %d", &num1, &num2);

    int sum = add(num1, num2);
    int difference = sub(num1, num2);
    int product = mul(num1, num2);
    int quotient = div(num1, num2);
```

```
    printf("Sum: %d\n", sum);
    printf("Difference: %d\n", difference);
    printf("Product: %d\n", product);
    printf("Quotient: %d\n", quotient);

    return
 0;
}
```

- **Step 4: Compile and Link**
    1. Use the gcc compiler to compile all the source files and link them into a single executable
    2. This command will create an executable file named operations.
- **Step 5: Execute the Program**

    Run the executable:

```
nivedita@nivedita-VirtualBox: ~/Nivedita
nivedita@nivedita-VirtualBox:~/Nivedita$ gcc main.c add.c sub.c mul.c div.c -o operation
nivedita@nivedita-VirtualBox:~/Nivedita$ ./operation
Enter two numbers: 10 5
Addition: 15
Difference: 5
Product: 50
Quotient: 2
nivedita@nivedita-VirtualBox:~/Nivedita$
```

Conclusion:

This experiment demonstrated the process of executing C programs in both single-file and multiple-file environments. Single-file execution is suitable for small, self-contained programs, while multiple files offer better organization, maintainability, and reusability for larger projects.

Questions:

1. Write meaning and use of GCC in Linux.

➔ **GCC (GNU Compiler Collection)** is a powerful compiler used for C, C++, and other programming languages. It's renowned for its portability, efficiency, and extensive optimization capabilities.

Key Features:

- **Cross-platform compatibility:** GCC can compile code for various architectures and operating systems, making it a versatile tool for developers.
- **Optimization:** GCC incorporates optimization techniques to generate efficient machine code, improving performance.
- **Standards compliance:** Adheres to the latest language standards for compatibility and portability.
- **Language support:** Supports a wide range of programming languages.
- **Debugging tools:** Includes built-in debugging tools like GDB for easier development.
- **Customization:** Can be customized through command-line options and configuration files.

Uses:

- **General-purpose software development:** Used to create various applications.
- **System software development:** Builds system-level tools and libraries.
- **Embedded systems:** Compiles code for devices with limited resources.
- **Research and development:** Utilized in academic and research projects.
- **Educational purposes:** Teaches programming concepts.

In summary, GCC is a valuable tool for developers working with C, C++, and other languages, offering flexibility, performance, and compatibility.

2. Write advantages and disadvantages of multiple C programming.

➔ **Advantages:**

- **Modularity:** Breaks down code into smaller, reusable components, improving organization and maintainability.
- **Reusability:** Functions can be used in multiple projects, reducing development time.
- **Collaboration:** Multiple developers can work on different parts of the code simultaneously.
- **Maintainability**: Easier to find, fix, and update specific parts of the code.
- **Scalability:** Handles larger projects more effectively by dividing code into manageable units.

Disadvantages:

- **Increased complexity:** Requires careful management of dependencies and header files.
- **Build process:** More complex build processes, especially for larger projects.
- **Potential for coupling:** Tight coupling between modules can make changes difficult.
- **Overhead:** Additional overhead of managing multiple files and dependencies.

In conclusion, multiple C programming offers significant advantages, especially for larger and more complex projects. However, it also introduces additional complexities that need to be carefully managed.