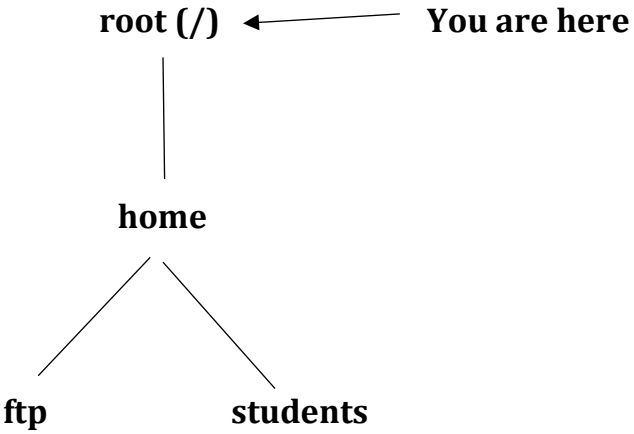
Q.NO. 1.

a. Given the tree diagram of the file structure, state appropriate Linux command(s) to answer the following questions. Commands from successive questions affect subsequent questions.



i.Move from "root" directory to "home" directory.

Ans:- cd /home

ii. Create a new file called "resit.txt" in your "home" directory.

Ans:- touch resit.txt

iii. Copy "resit.txt" from "home" directory to "students" directory.

Ans:- cp resit.txt /students

iv. Move from "home" directory to "students" directory.

Ans:- cd /students

v. Copy "resit.txt" to "final.txt".

Ans:- cp resit.txt final.txt

vi. List all files in "students" directory.

Ans:- ls

b. Write a C program that asks the user to enter a file name and delete a filename using Linux command according to the input file name.

Ans:-

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

int main() {
    char filename[100];

    printf("Enter the file name to delete: ");
    scanf("%s", filename);

    if (remove(filename) == 0) {
        printf("File '%s' deleted successfully.\n", filename);
    } else {
        printf("Unable to delete the file '%s'.\n", filename);
    }

    return 0;
}
```

Q.NO. 2.

a. Define the following terms:

i. Strings **ii. Call by value**

iii. Variables **iv. Datatypes**

v. Function definition

Ans:-

i. Strings:- Strings are sequences of characters, typically used to represent text in programming. They can consist of letters, numbers, symbols, or any combination thereof. In many programming languages, strings are enclosed in quotation marks (e.g., "Hello, World!").

ii. Call by Value:- Call by value is a method of passing arguments to a function in programming. When you pass a value to a function using call by value, a copy of the value is created, and the function operates on this copy. Any changes made to the parameter within the function do not affect the original value outside of the function. This method is commonly used in languages like C and Java.

iii. Variables:- Variables are symbols or identifiers used to store data values in a program. They provide a way to name and reference data in memory. Variables can hold various types of data, such as numbers, text, and more complex data structures. They are a fundamental concept in programming and are used to store and manipulate data during program execution.

iv. Datatypes:- Datatypes are a classification system used in programming to categorize and specify the type of data that a variable can hold or a function can return. Different programming languages support various datatypes, including integers, floating-point numbers, strings, booleans, arrays, and more. Datatypes determine the kind of operations that can be performed on the data and how it is stored in memory.

v. Function Definition:- A function definition is a part of a program where a specific set of instructions is written to define how a function should behave. It includes the function's name, its parameters (if any), and the code that gets executed when the function is called. Function definitions are used to encapsulate a block of code so that it can be reused multiple times in a program, promoting code modularity and reusability. Functions are a fundamental building block of structured programming.

b. Write a C program to find how many consonants and vowels in the string below.

char name []="GOOD MORNING";

Ans:-

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

int main() {
    char name[] = "GOOD MORNING";
    int vowels = 0;
    int consonants = 0;

    for (int i = 0; name[i] != '\0'; i++) {
        name[i] = toupper(name[i]);
    }

    for (int i = 0; name[i] != '\0'; i++) {
        if (name[i] >= 'A' && name[i] <= 'Z') {
            if (name[i] == 'A' || name[i] == 'E' || name[i] == 'I' || name[i] == 'O' || name[i] == 'U') {
                vowels++;
            } else {
                consonants++;
            }
        }
    }

    printf("Number of vowels: %d\n", vowels);
    printf("Number of consonants: %d\n", consonants);

    return 0;
}

```

Q.NO. 3.

a. What is the result of executing the following codes?

i. int x = 1, total = 0, y;

while (x <= 10)

{

```
    y = x * x;
    printf( "%d\n", y );
    total += y;
    ++x;
```

}

printf("Total is %d\n", total);

Ans:-

1

4

9

16

25

36

49

64

81

100

Total is 385

ii. int x = 6;

int y = 5;

```
if (x>y) {
        printf( "%d is larger than %d\n", x,y );
}
if ( x < y ) {
        printf( "%d is smaller than %d\n", x,y );
}
if ( x == y ) {
        printf( "These numbers are equal\n" );
}
```

Ans:- 6 is larger than 5.

b. for (int index=0;index < 10; index++)

```
{
        sampleArray[index]=3*index;
        printf("%d ",sampleArray[index]);
}
```

Convert the for loop statement above to do..while loop.

Ans:-

```
int index = 0;
do {
   sampleArray[index] = 3 * index;
   printf("%d ", sampleArray[index]);
   index++;
} while (index < 10);
```

c. Explain THREE (3) types of programming errors

Ans:- Here are three types of programming errors explained briefly:

1. Syntax Errors: Syntax errors are the most common type of programming errors. They occur when the code violates the rules of the programming language's syntax. This could be a missing semicolon, a typo, or incorrect indentation. These errors prevent the code from running at all.

2. Runtime Errors: Runtime errors occur when the code is syntactically correct but encounters an issue while it's running. These can include division by zero, trying to access an item in an array that doesn't exist, or using a variable before it's been initialized. Runtime errors can cause the program to crash or produce unexpected results.

3. Logical Errors: Logical errors are the trickiest to spot. They occur when the code runs without any syntax or runtime errors, but it doesn't produce the expected output. These errors are often the result of flawed logic or incorrect algorithms in the code. Debugging logical errors can be challenging because the code appears to be correct, but it doesn't perform as intended.

d. Write a program using nested for loop to produce the following output:

10

9 8

8 7 6

7 6 5 4

6 5 4 3 2

Ans:-

```c
#include <stdio.h>

int main() {
    int n = 10;

    for (int i = 1; i <= 5; i++) {
        for (int j = 1; j <= i; j++) {
            printf("%d\t", n);
            n--;
        }

        n = 10 - i;
        printf("\n");
    }

    return 0;
}
```

Q.NO. 4.

Write a menu driven program in C to get two integers from the input. Then the program performs the following task based on the option:

If the user enters 1, the program prints the addition of two integers.

If the user enters 2, the program prints the subtraction of two integers.

If the user enters 3, the program prints the multiplication of two integers.

If the user enters 4, the program prints the quotient of two integers.

If the user enters 5, the program will exit from the program

Sample output:

Enter First Number: 6

Enter Second Number: 4

1. Addition

2. Subtraction

3. Multiplication

4. Quotient

5. Exit

Option : 1

6 + 4 = 10

1. Addition

2. Subtraction

3. Multiplication

4. Quotient

5. Exit

Option : 2

6 – 4 = 2

1.Addition

2. Subtraction

3. Multiplication

4. Quotient

5. Exit

Option : 5

End of Program

Ans

```c
#include <stdio.h>

int main() {
    int num1, num2, option, result;

    while (1) {
        printf("Enter First Number: ");
        scanf("%d", &num1);

        printf("Enter Second Number: ");
        scanf("%d", &num2);

        printf("1. Addition\n");
        printf("2. Subtraction\n");
        printf("3. Multiplication\n");
        printf("4. Quotient\n");
        printf("5. Exit\n");
        printf("Option : ");
        scanf("%d", &option);
```

```c
        switch (option) {
            case 1:
                result = num1 + num2;
                printf("%d + %d = %d\n", num1, num2, result);
                break;
            case 2:
                result = num1 - num2;
                printf("%d - %d = %d\n", num1, num2, result);
                break;
            case 3:
                result = num1 * num2;
                printf("%d * %d = %d\n", num1, num2, result);
                break;
            case 4:
                if (num2 == 0) {
                    printf("Cannot divide by zero!\n");
                } else {
                    result = num1 / num2;
                    printf("%d / %d = %d\n", num1, num2, result);
                }
                break;
            case 5:
                printf("End of Program\n");
                return 0;
            default:
                printf("Invalid option! Please choose a valid option.\n");
        }
    }

    return 0;
}

```