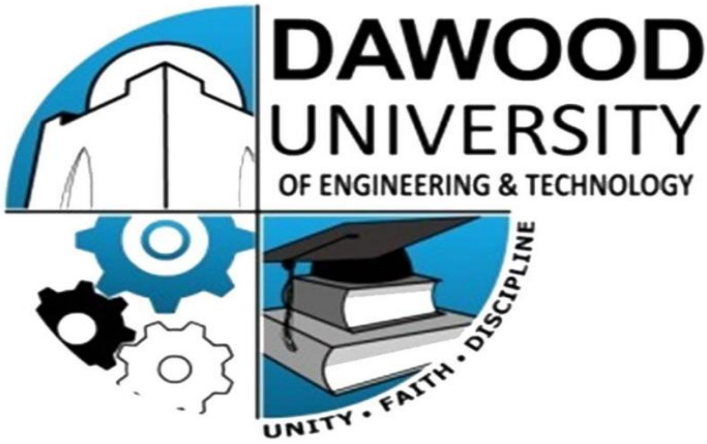

NAME : SAFI AHMED

ROLL NO : 22F-BSCS-35

SUBJECT: PROGRAMMING FUNDAMENTAL

SUBMITTED TO : ENGR. SOFIA HAJANO

1. Discuss the key differences between C and C++ programming languages. Provide examples and explanations for at least three major differences.

Instructions:

- Compare and contrast the syntax and features of C and C++.
- Explain how C++ extends C by incorporating additional features.
- Highlight the advantages and potential use cases for each language.
- Provide code snippets or examples to support your explanations.

C and C++ are both powerful programming languages, but they differ in terms of syntax, features, and use cases. Here are three major differences between C and C++, along with examples and explanations:

C Programming Language	C++ Programming Language
Object-Oriented Programming (OOP) Support:	
C is a procedural programming language that focuses on functions and structures. It does not have built-in support for object-oriented programming.	C++ extends C by incorporating object-oriented programming features, such as classes, objects, inheritance, polymorphism, and encapsulation.
In C, you can use structures to group related data together, but they do not have member functions or inherent behavior.	With C++, you can define classes that encapsulate data and functions, allowing for better code organization and reusability.

Example (C):

```c
// Structure in C
struct Rectangle {
    int length;
    int width;
};
```

Example (C++):

```cpp
// Class in C++
class Rectangle {
private:
    int length;
    int width;

public:
    Rectangle(int l, int w) : length(l), width(w) {}
    int area() { return length * width; }
};
```

	Advantage: C++'s support for OOP enables better code organization, modularity, and code reuse. It is well-suited for building complex software systems, large-scale applications, and frameworks.
Standard Template Library (STL):	
C does not provide a built-in library for data structures and algorithms. Developers often need to implement them manually or rely on third-party libraries.	C++ includes the Standard Template Library (STL), which provides a rich set of pre-defined containers (e.g., vector, list, map) and algorithms (e.g., sorting, searching) that can be used out of the box. The STL simplifies common programming tasks by offering efficient and reusable components.

Example (C++):

```cpp
// Using STL vector in C++
#include <vector>
#include <iostream>

int main() {
    std::vector<int> numbers {1, 2, 3, 4, 5};
    for (int num : numbers) {
        std::cout << num << " ";
    }
    return 0;
}
```

	Advantage: The STL in C++ enhances productivity by providing a wide range of data structures and algorithms that are efficient and reliable. It simplifies common programming tasks, such as managing collections and performing operations on them.
Memory Management	
C requires manual memory management using functions like malloc() and free(). Developers need to explicitly allocate and deallocate memory.	C++ introduces automatic memory management through constructors and destructors.
If not done correctly, it can lead to memory leaks or access violations.	It provides dynamic memory allocation with new and delete keywords. Additionally, it offers smart pointers (e.g., std::unique_ptr, std::shared_ptr) that automatically handle memory deallocation.

Example (C):

```c
// Manual memory management in C
int* numPtr = (int*)malloc(sizeof(int));  // Allocating memory
*numPtr = 42;                             // Assigning value
free(numPtr);                             // Deallocating memory
```

Example (C++):

```cpp
// Dynamic memory allocation in C++
int* numPtr = new int;    // Allocating memory
*numPtr = 42;             // Assigning value
delete numPtr;            // Deallocating memory
```

	Advantage: C++'s memory management features reduce the chances of memory leaks and make code more robust.
	Smart pointers in C++ help in automatically releasing memory, improving code reliability and reducing manual memory management errors.
C is a low-level language that provides fine-grained control over hardware, making it suitable for system programming, embedded systems, and performance-critical applications.	C++ extends C by incorporating additional features such as OOP, the STL, and automatic memory management. This makes it well-suited for building large-scale applications, frameworks, and software systems that require modularity, reusability, and a higher level of abstraction.
Function Overloading:	
C does not support function overloading, which means functions must have unique names even if they perform similar tasks.	C++ allows function overloading, which means multiple functions with the same name but different parameters can coexist. The appropriate function is called based on the arguments provided.

Example (C++):

```cpp
// Function overloading in C++
int add(int a, int b) {
    return a + b;
}

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

Function Syntax:	
In C, functions are defined using the following syntax:	C++ follows the same syntax for function definitions as C, but it also supports function overloading, allowing multiple functions with the same name but different parameter lists.

Example (C):

```
return_type function_name(parameter_list) {
    // Function body
}
```

Example (C++):

```
return_type function_name(parameter_list) {
    // Function body
}

// Function overloading in C++
return_type function_name(parameter_list1) {
    // Function body
}

return_type function_name(parameter_list2) {
    // Function body
}
```

Standard Input/Output:	
In C, the standard input/output operations are performed using functions like printf() and scanf().	C++ introduces the concept of input/output streams, and the standard input/output is handled using cin and cout objects from the iostream library.

Example (C):

```c
#include <stdio.h>

int main() {
    int num;
    printf("Enter a number: ");
    scanf("%d", &num);
    printf("You entered: %d", num);
    return 0;
}
```

Example (C++):

```cpp
#include <iostream>
using namespace std ;

int main() {
    int num;
    cout << "Enter a number: "<<endl;
    cin >> num;
    cout << "You entered: " << num;
    return 0;
}

```

Exception Handling:	
C does not have built-in exception handling mechanisms. Error handling is typically done using error codes or return values.	C++ provides exception handling through try, catch, and throw statements. Exceptions can be thrown when an error occurs, and caught in an appropriate catch block for handling.

Example (C++):

```cpp
try {
    // Code that may throw an exception
    throw ExceptionType(); // Throw an exception of a specific type
}
catch (ExceptionType& e) {
    // Exception handling code
}
```

C Advantages and Use Cases:

Efficiency:
C allows for low-level programming and direct hardware manipulation, making it efficient in terms of memory usage and execution speed. It is often used in embedded systems and operating systems where efficiency is crucial.
Portability:
C programs are highly portable and can be easily compiled and executed on different platforms with minimal changes. This makes it suitable for developing cross-platform software.
Systems Programming:
C is widely used for systems programming tasks, such as device drivers, kernels, and operating systems, where direct hardware interaction and control are required.
Performance-Critical Applications:
C is preferred for performance-critical applications, such as real-time systems and high-performance computing, where every CPU cycle counts.

Efficiency:

```c
#include <stdio.h>

int main() {
    int num1 = 10;
    int num2 = 20;
    int sum = num1 + num2;
    printf("Sum: %d\n", sum);
    return 0;
}
```

Portability:

```c
#include <stdio.h>

int main() {
    printf("Hello, World!\n");
    return 0;
}
```

Systems Programming:

```c
#include <stdio.h>

int main() {
    FILE* file = fopen("data.txt", "r");
    if (file != NULL) {
        char buffer[256];
        while (fgets(buffer, sizeof(buffer), file) != NULL) {
            printf("%s", buffer);
        }
        fclose(file);
    }
    return 0;
}
```

C++ Advantages and Use Cases:

Object-Oriented Programming (OOP):
C++ introduces OOP concepts, allowing for better code organization, reusability, and modularity. It enables the creation of complex software systems and large-scale applications.
Standard Template Library (STL):
C++ provides the STL, a powerful library that includes pre-defined containers, algorithms, and utilities. The STL simplifies common programming tasks, such as managing collections, searching, sorting, and data manipulation.
GUI Applications:
C++ is often used for developing graphical user interface (GUI) applications. Libraries like Qt and wxWidgets provide robust tools and frameworks for building desktop applications.

Game Development:
C++ is widely used in the game development industry due to its performance, low-level access, and support for OOP. Game engines like Unreal Engine and Unity are built using C++ and offer extensive C++ APIs.
High-Level Abstraction:
C++ allows both low-level and high-level programming, providing a balance between performance and abstraction. It is suitable for projects that require a mix of high-level features and low-level control.

Object-Oriented Programming (OOP):

Welcome project2.cpp

project2.cpp > ...

```cpp
#include <iostream>
using namespace std;

class Circle {
private:
    double radius;

public:
    Circle(double r) : radius(r) {}

    double calculateArea() {
        return 3.14159 * radius * radius;
    }
};

int main() {
    Circle circle(5.0);
    double area = circle.calculateArea();
    cout << "Area: " << area << endl;
    return 0;
}
```

Standard Template Library (STL):

```cpp
#include <iostream>
#include <vector>
using namespace std;

int main() {
    vector<int> numbers {1, 2, 3, 4, 5};
    for (int num : numbers) {
        cout << num << " ";
    }
    cout << endl;
    return 0;
}
```

GUI Applications:

```cpp
#include <iostream>
#include <QtWidgets/QApplication>
#include <QtWidgets/QPushButton>

int main(int argc, char** argv) {
    QApplication app(argc, argv);
    QPushButton button("Click me!");
    button.show();
    return app.exec();
}
```

1. Write a C++ program that prompts the user to enter any three integer numbers and displays the number in descending order.

SOURCE CODE

Welcome | PF_Assignment.cpp

PF_Assignment.cpp > main()

```cpp
#include<iostream>
using namespace std;
int main(){
    int num1,num2,num3;
    cout<<"Enter the First number: "<<endl;
    cin >>num1;
    cout<<"Enter the Second number: "<<endl;
    cin >>num2;
    cout<<"Enter the Third number: "<<endl;
    cin >>num3;
    if(num1>num2 && num1>num3)
    {
        if( num2>num3)
        {
            cout<<num1<<","<<num2<<","<<num3<<endl;
        }
        else{
            cout<<num1<<","<<num3<<","<<num2<<endl;
        }
    }
    if(num2>num1 && num2>num3)
    {
        if( num1>num3)
        {
            cout<<num2<<","<<num1<<","<<num3<<endl;
        }
        else{
            cout<<num2<<","<<num3<<","<<num1<<endl;
        }
    }
    if(num3>num1 && num3>num2)
    {
        if(num1>num2)
        {
            cout<<num3<<","<<num1<<","<<num2<<endl;
        }
        else{
            cout<<num3<<","<<num2<<","<<num1<<endl;
        }
    }
return 0;
}
```

OUTPUT

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS C:\Users\Mohit Computers\Desktop\16 JUNE> cd "c:\Users\Mohit Computers\Desktop\16 JUNE\" ; if ($?) { g++ PF_Assignment.cpp -o PF_Assignment } ; if ($?) { .\PF_Assignment }
Enter the First number:
24
Enter the Second number:
98
Enter the Third number:
57
98,57,24
PS C:\Users\Mohit Computers\Desktop\16 JUNE>
```

3. The formulas for calculating Body Mass Index (BMI) are

$$BMI = \frac{weightInPounds \times 703}{heightInInches \times heightInInches}$$

or

$$BMI = \frac{weightInKilograms \times 703}{heightInMeters \times heightInMeters}$$

Create a BMI calculator application in C++ that reads the user's weight in pounds and height in inches (or, if you prefer, the user's weight in kilograms and height in meters), then calculates and displays the user's body mass index

.

SOURCE CODE

Welcome | PF_Assigment.cpp

PF_Assigment.cpp > main()

```cpp
#include <iostream>
using namespace std;

int main() {
    double weight, height;

    //cout << "Enter your weight in kilograms: ";
    cout << "Enter your weight in pounds: ";
    cin >> weight;

    //cout << "Enter your height in meters: ";
    cout << "Enter your height in inches: ";
    cin >> height;


    double bmi = (weight / (height * height)) * 703;


    cout << "Your BMI is: " << bmi << endl;

    return 0;
}
```

OUTPUT

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

PS C:\Users\Mohit Computers\Desktop\16 JUNE> cd "c:\Users\Mohit Computers\Desktop\16 JUNE\" ; if ($?) { g++ PF_Assigment.cpp -o PF_Assigment } ; if ($?) { .\PF_Assigment }
Enter your weight in pounds: 110.23
Enter your height in inches: 69
Your BMI is: 16.2763
PS C:\Users\Mohit Computers\Desktop\16 JUNE>
```