# C++ Basics Programming Assignment

## Section 1: Basic Syntax and Data Types (15 points)

### 1. (3 pts) Explain the purpose of the main() function in a C++ program

The `main()` function serves as the starting point for execution in every C++ program. When a user runs an executable, the operating system loads the program into memory and begins execution from this function. It also returns an integer value to the operating system to indicate whether the program ended successfully.

### 2. (4 pts) List four fundamental data types in C++ and briefly describe each

- `int`: Used to store whole numbers (integers).
- `char`: Used to store a single character.
- `bool`: Represents a Boolean value, which can be either true or false.
- `double`: Used to store double-precision floating-point numbers (real numbers with decimals).

### 4. (4 pts) Explain the difference between float and double. Include one situation where double is preferred

- `float` represents a single-precision floating-point number, while `double` represents a double-precision floating-point number. The exact size depends on the implementation, but `double` typically offers more precision than `float`.
- `double` is preferred when **more accuracy** is required for calculations. Additionally, default floating-point literals in C++ (e.g., `3.14`) are treated as `double` by default, making it a standard choice for general floating-point arithmetic.

## Section 2: Control Structures (20 points)

### 6. (5 pts) Explain the difference between a while loop and a do-while loop. Include a short code snippet for each

- **`while` loop:** This is an **entry-controlled** loop. The condition is evaluated *before* the loop body executes. If the condition is false initially, the loop body never runs.

```
int i = 0;
while (i < 5) {
    cout << i << " ";
    i++;
}
```

- **`do-while` loop:** This is an **exit-controlled** loop. The loop body executes first, and then the condition is evaluated. This guarantees that the loop body runs **at least once**, regardless of whether the condition is true or false.

```
int i = 0;
do {
    cout << i << " ";
    i++;
} while (i < 5);
```

## Section 3: Functions (20 points)

### 9. (4 pts) Write a function declaration and definition for a function `calculateArea` that takes two double parameters (length and width) and returns the area

```
// Function Declaration
double calculateArea(double length, double width);

// Function Definition
double calculateArea(double length, double width) {
    return length * width;
}
```

**10. (4 pts) Explain function overloading in C++. Provide a small example with at least two overloaded functions**

- Function overloading allows multiple functions to share the same name as long as they have different parameter lists.
- The compiler determines which function to call based on the arguments passed.

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

// Function 1: Calculates area of a square (1 parameter)
int area(int side) {
return side \* side;
}

// Function 2: Calculates area of a rectangle (2 parameters)
int area(int length, int width) {
return length \* width;
}

int main() {
cout << "Square: " << area(5) << endl; // Calls Function 1
cout << "Rectangle: " << area(5, 10) << endl; // Calls Function 2
return 0;
}
```

**11. (4 pts) Explain the difference between pass-by-value and pass-by-reference. When should pass-by-reference be used?**

Difference:

- Pass-by-value: The function receives a copy of the variable. Changes made to the parameter inside the function do not affect the original variable in the calling function. This is the default behavior for fundamental types like int.
- Pass-by-reference: The function receives a reference (an alias) to the original variable's memory address. Changes made to the parameter inside the function directly modify the original variable.

When to use Pass-by-Reference:

- Modifying Data: When the function needs to change the value of the argument (e.g., a swap function).
- Efficiency: When passing large data structures (like a large vector or string ) to avoid the performance cost of copying the entire object.

**12. (4 pts) What are default arguments in C++? Write a function example that uses a default argument**

- Default arguments are values assigned to function parameters in the function declaration.
- If the caller omits the argument for that parameter, the default value is used automatically.

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

// Function with a default argument for 'b'
void displaySum(int a, int b = 10) {
    cout << "Sum: " << a + b << endl;
}

int main() {
    displaySum(5);      // Uses default b = 10. Output: 15
    displaySum(5, 20);  // Overrides default.   Output: 25
    return 0;
}
```

## Section 4: Object-Oriented Programming (20 points)

**15. (4 pts) Explain encapsulation in C++. How is it implemented in a class?**

- Explanation:
  - Encapsulation is the OOP concept of bundling data (variables) and the methods (functions) that operate on that data into a single unit, known as a class.
  - Crucially, it also involves data hiding: restricting direct access to some of an object's components to prevent accidental modification or misuse.
- Implementation:
  - Data is typically marked as private so it cannot be accessed directly from outside the class.
  - Methods (getters and setters) are marked as public to provide a controlled interface for reading or modifying that private data.

**16. (4 pts) Explain the difference between public, private, and protected access specifiers**

- `Public`: Members declared as public are accessible from anywhere in the program (inside the class, inside derived classes, and from main() or other functions).
- `Private`: Members declared as private are accessible only within the class that defines them. They are hidden from derived classes and the outside world.
- `Protected`: Members declared as protected are accessible within the class that defines them and within any classes that inherit from it (derived classes). However, they remain inaccessible to the outside world (like main()).

**17. (6 pts) Explain inheritance in C++. Write a simple example where a class Square inherits from Rectangle**

Inheritance is a mechanism where a new class (called Child Class) acquires the properties and behaviors of an existing class (called Parent Class). This promotes code reusability.

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

class Rectangle {
protected:
    double length;
    double width;
public:
    void setValues(double l, double w) {
        length = l;
        width = w;
    }
    double getArea() {
        return length * width;
    }
};

class Square : public Rectangle {
public:
    void setSide(double side) {
        setValues(side, side);
    }
};

int main() {
    Square s;
    s.setSide(4.0);

    cout << "Area of Square: " << s.getArea() << endl;

    return 0;
}
```

## Section 5: Memory Management and Pointers (15 points)

**19. (5 pts) What is a memory leak? Explain two ways to prevent memory leaks in C++**

A memory leak occurs when a program allocates memory on the heap (using `new`) but fails to deallocate it (using `delete`) before the program terminates or before the pointer to that memory is lost. This causes the program to consume more memory than necessary, potentially leading to performance issues or crashes.

Two ways to prevent memory leaks:

- Always Pair `new` with `delete`: For every manual allocation using new, ensure there is a corresponding `delete` in the code path where the memory is no longer needed.
- Use Smart Pointers or Containers:
  - `std::vector`: Automatically manages memory allocation and deallocation, growing and shrinking as needed.
  - Smart Pointers: Use `std::unique_ptr` or `std::shared_ptr`, which automatically delete the associated memory when the pointer goes out of scope.

## Section 6: Error Handling and const Usage (10 points)

**21. (5 pts) Explain exception handling in C++. Write a simple try-catch example that handles division by zero**

Exception handling provides a way to transfer control from one part of a program to another when an unexpected error occurs (an "exception"). It prevents the program from crashing abruptly.

- `try`: Wraps the code that might generate an error.
- `throw`: Used to signal that an error has occurred.
- `catch`: Captures the exception thrown and handles it (e.g., prints an error message).

**22. (5 pts) Explain the purpose of the const keyword. Write a short C++ program demonstrating a constant variable and a function parameter declared as const**

The `const` keyword (short for "constant") specifies that a variable's value cannot be modified after it is initialized.