# Evaluation Report

Grade: 8

Similarity Score: 0.90

1. Difference between == and .equals() in C++ In C++, the == operator is used to compare primitive data types and objects. However, .equals() is not a standard method in C++ (unlike in Java, where equals() is a method for comparing objects). In C++, if you want to compare objects, you need to overload the == operator for your class. Example for ==: int a = 5; int b = 5; if (a == b) { // This is true because both a and b have the same value. } Example of overloading == for a custom class: class MyClass { public: int value; bool operator==(const MyClass& other) const { return this ->value == other.value; } }; MyClass obj1{5}; MyClass obj2{5}; if (obj1 == obj2) { // This will be true because we've overloaded the == operator to compare the value fields. } 2. Memory Management in C++ C++ handles memory management through manual allocation and deallocation using pointers, new, and delete operators. Pointers: Variables that hold the address of another variable. They are essential for dynamic memory management. new operator: Allocates memory on the heap for a variable and returns a pointer to it. delete operator: Frees memory allocated on the heap, preventing memory leaks. Example: int\* ptr = new int; // Allocate memory for an int on the heap \*ptr = 10; // Assign value to the allocated memory delete ptr; // Free the allocated memory int\* arr = new int[10]; // Allocate memory for an array of 10 ints on the heap delete[] arr; // Free the allocated memory for the array 3. Purpose of the const Keyword in C++ The const keyword specifies that a variable's value cannot be modified after initialization. It can be used in various contexts, such as with variables, pointers, member functions, and function parameters. Example: const int x = 10; // x is a constant and cannot be modified void printValue(const int& value) { // value is a reference to a constant int and cannot be modified within this function std::cout << value << std::endl; } class MyClass { public: void myFunction() const { // This member function cannot modify any member variables of the class } }; 4. Function Overloading vs Function Overriding Function Overloading: Allows multiple functions with the same name but different parameter lists within the same scope. The compiler differentiates these functions by their signature (number and type of parameters). Example: void print(int i) { std::cout << i << std::endl; } void print(double d) { std::cout << d << std::endl; } void print(std::string s) { std::cout << s << std::endl; } Function Overriding: Allows a subclass to provide a specific implementation of a method that is already defined in its superclass. The method in the subclass has the same signature as the method in the superclass. Requires the virtual keyword in the base class method. Example : class Base { public: virtual void display() { std::cout << "Display from Base" << std::endl; } }; class Derived : public Base { public: void display() override { std::cout << "Display from Derived" << std::endl; } }; Base\* b = new Derived(); b->display(); // Outputs "Display from Derived" 5. Significance of the virtual Keyword The virtual keyword is used to indicate that a member function can be overridden in derived classes. It enables polymorphism, allowing the correct method to be called based on the object's type at runtime. Inheritance and Polymorphism: Inheritance: Allows a class (derived class) to inherit properties and behaviors (methods) from another class (base class). Polymorphism: Allows a function to behave differently based on the object that is calling it. Achieved through virtual functions. Example: class Animal { public: virtual void makeSound() const { std::cout << "Some generic animal sound" << std::endl; } }; class Dog : public Animal { public: void makeSound() const override { std::cout << "Woof" << std::endl; } }; class Cat : public Animal { public: void makeSound() const override { std::cout << "Meow" << std::endl; } }; void describeSound(const Animal& animal) { animal.makeSound(); // Calls the appropriate makeSound function based on the object type } Animal a; Dog d; Cat c; describeSound(a); // Outputs "Some generic animal sound" describeSound(d); // Outputs "Woof" describeSound(c); // Outputs "Meow"

1. Difference between "==" and ".equals()" in C++ In C++, there is no .equals() method as there is in languages like Java. Instead, the == operator is commonly overloaded to compare objects. Here’s a detailed explanation: == Operator: This operator is used to compare the values of two variables. For primitive data types (like int, char, etc.), it checks for value equality. For objects, it can be overloaded to provide custom comparison logic. .equals() Method: While C++ does not natively provide an .equals() method, similar functionality can be achieved through member functions. For example, you can define an equals method in your class to compare objects. Example: class MyClass { public: int data; bool operator==(const MyClass &other) const { return this->data == other.data; } bool equals(const MyClass &other) const { return this->data == other.data; } }; MyClass obj1, obj2; obj1.data = 5; obj2.data = 5; if (obj1 == obj2) { // Uses the overloaded == operator } if (obj1.equals(obj2)) { // Uses the equals method } 2. Memory Management in C++: Pointers, new, and delete Operators C++ provides explicit control over memory management through pointers and dynamic memory allocation. Pointers: Pointers are variables that store memory addresses. They are used to directly access and manipulate memory. int x = 10; int \*ptr = &x; // ptr holds the address of x new Operator: This operator allocates memory on the heap and returns a pointer to the beginning of the allocated memory. It is used for dynamic memory allocation. int \*p = new int; // dynamically allocate memory for an int \*p = 20; delete Operator: This operator deallocates memory that was previously allocated with new, preventing memory leaks. delete p; // free the memory allocated for the int For arrays, new[] and delete[] are used: int \*arr = new int[10]; // dynamically allocate memory for an array of 10 ints delete[] arr; // deallocate the array memory 3. Purpose of the const Keyword in C++ The const keyword is used to define variables or parameters whose value cannot be changed after initialization. It can be applied to variables, pointers, function parameters, and member functions. Examples: Const Variable: const int MAX = 100; Const Pointer: const int \*ptr = &MAX; // pointer to a const int int \*const ptr2 = &x; // const pointer to an int Const Function Parameter: void print(const int value) { // value cannot be modified inside this function } Const Member Function: class MyClass { public: void display() const { // this function cannot modify any member variables } }; 4. Function Overloading vs Function Overriding in C++ Function Overloading: This is a feature where multiple functions can have the same name but different parameters (different type or number of parameters). It is resolved at compile time (compile-time polymorphism). Example: void display(int i) { // implementation for int } void display(double d) { // implementation for double } Function Overriding: This occurs when a derived class has a definition for one of the member functions of the base class. That base function is said to be overridden. It is resolved at runtime (runtime polymorphism). Example: class Base { public: virtual void show() { // base class implementation } }; class Derived : public Base { public: void show() override { // derived class implementation } }; 5. Significance of the virtual Keyword in C++ The virtual keyword is used to declare a member function in the base class that can be overridden in a derived class. It enables runtime polymorphism, allowing the program to decide at runtime which function to call based on the type of object being referenced. Example: class Base { public: virtual void display() { cout << "Base display" << endl; } }; class Derived : public Base { public: void display() override { cout << "Derived display" << endl; } }; Base \*b = new Derived(); b->display(); // calls Derived's display() method due to virtual keyword The virtual keyword ensures that the correct function is called for an object, regardless of the type of reference (or pointer) used for the function call, enabling dynamic (runtime) polymorphism.

## Explanations for Highlights

Words missing in the student's submission are highlighted in red.

Words added by the student are highlighted in yellow.

## Summary

The student has received a grade of 8 with a similarity score of 0.90. Major strengths and weaknesses identified in the comparison.