Object Oriented Programming using C++

FUNCTION OVERLOADING

- Overloading Concept: Overloading refers to using the same name for different purposes. In C++, this applies to functions, where the same function name can be used for multiple functions performing different tasks.

- Purpose: Function overloading is a key feature of C++ that supports compile-time polymorphism. It enhances flexibility and convenience by allowing the use of the same function name for various operations.

- Function Polymorphism: This allows a single function name to be associated with different tasks. The exact function that gets called is determined by the number or type of arguments passed.

- How It Works:

  - Each overloaded function must differ either in the type of its parameters or the number of its parameters.

  - The compiler determines which version of the function to call based on the arguments provided.

Example of Function Overloading

```
#include <iostream>
using namespace std;
// Function to add two integers
int add(int a, int b) {
    return a + b;
}
// Function to add three integers
int add(int a, int b, int c) {
    return a + b + c;
}
// Function to add two doubles
double add(double a, double b) {
    return a + b;
}
int main() {
    cout << "Sum of 2 and 3: " << add(2, 3) << endl;        // Calls int add(int, int)
    cout << "Sum of 2, 3, and 4: " << add(2, 3, 4) << endl; // Calls int add(int, int, int)
    cout << "Sum of 2.5 and 3.5: " << add(2.5, 3.5) << endl; // Calls double add(double, double)
    return 0;
}
```

Function Overloading Restrictions in C++

Cannot Overload by Return Type Alone:

Functions cannot be overloaded based solely on their return type. This means that if two functions differ only by their return type, they cannot be considered overloaded.

Example:

```
int myfun(int i);

float myfun(int i); // Error: Cannot overload based on return type alone
```

In this case, the two `myfun` functions appear to differ because one returns an `int` and the other a `float`, but this difference is insufficient for overloading.

Pointer and Array Parameters: The compiler treats pointers and arrays as the same type when it comes to function parameters, which can lead to errors if you attempt to overload functions that differ only by these types.

- Example:

```
void f(int *p);

void f(int p[]); // Error: Cannot overload, *p is the same as p[]
```

Although the function prototypes `void f(int *p);` and `void f(int p[]);` appear different, the compiler treats them as the same because `*p` is equivalent to `p[]`.

Key Points:

- Overloading requires a difference in the number or type of parameters, not just the return type.

- Be mindful of pointer and array types, as they are considered equivalent in function parameters.