Function Overloading In C++ :

Function overloading is when more than one function has the same name but different signatures. In this case, the function call decides which overloaded function to run.

Function Overloading can be achieved in the following ways:

- A different number of parameters
- Different data types of parameters

How Does Function Overloading Work?

Function overloading in C++ allows multiple functions with the same name but different parameters to coexist. The compiler differentiates these functions based on the number, types, or order of parameters. When an overloaded function is called, the compiler determines which version to execute by matching the call's arguments to the function's parameters. This feature enables more readable and intuitive code, allowing functions to handle different types of data without needing distinct names for each variant.

Why is Function Overloading in C++ Used?

Function overloading is used in C++ to enhance code readability and maintainability. It allows programmers to use the same function name for different operations, provided they differ in parameter type or count. This reduces the need for multiple function names, simplifying the interface and making the code more intuitive. Overloading also supports polymorphism, enabling functions to operate on various data types, thus promoting code reuse and reducing redundancy.

Rules of Function Overloading in C++

- **Different Parameter Types**: Functions must have different parameter types.
- **Number of Parameters**: Functions can have a different number of parameters.

- **Parameter Order**: Functions can have parameters in different orders.
- **Return Type Ignored**: The return type is not considered for overloading. Functions cannot be overloaded solely on different return types.
- **Default Arguments**: Functions with default arguments must be considered carefully, as they might conflict with other overloaded versions.
- **Scope Considerations**: Function overloading is applied within the same scope. Functions in different scopes (like base and derived classes) do not affect each other.
- **Const and Non-Const Parameters**: Functions can be overloaded based on const and non-const parameters.
- **Pointer and Reference Parameters**: Functions can be overloaded with pointer and reference parameters.
- **Ellipsis (…) Parameters**: Functions can be overloaded if they differ by including an ellipsis (…) parameter.

Which Function to Run When an Overloaded Function is Called?

When an overloaded function is called, the compiler determines the most appropriate function definition to use, by comparing the number of arguments and argument types you have used to call the function. This process of selecting the most appropriate overloaded function is called **overload resolution.**
The steps of overload resolution are:

- Find suitable functions via name lookup. These functions are called candidate functions.
- Remove invalid candidate functions from the list. The left out functions are called viable functions. A candidate function becomes invalid when:
    - The passed argument count does not match the parameter list.
    - Passed arguments types do not match the function parameter.
- Viable functions are then ranked.
    - Ranking order: Exactly match parameters > parameters matched after standard conversions > parameters matched after user-defined conversions
- If the best match is found from the viable function list, then that function is executed; else the compiler returns an error.

Note: Function overloading is independent of the return type.

Types of Function Overloading in C++

There are two types of function overloading in C++.

1. **Compile-Time Overloading:** Compile-time overloading, also known as static polymorphism, occurs when the function to be invoked is determined at compile time. This is achieved through function overloading and operator overloading, allowing multiple functions with the same name but different parameters to coexist.
2. **Runtime Overloading:** Runtime overloading, also known as dynamic polymorphism, involves determining the function to be invoked at runtime. This is typically achieved through virtual functions in base and derived classes, enabling function overriding where derived classes provide specific implementations of functions declared in the base class.

Advantages of Function Overloading in C++

Some of the benefits of function overloading are:

- Programs execution becomes faster.
- Smooth and simple code flow.
- Code maintenance becomes easier.
- Improves code readability.
- Saves memory space.
- Code reusability achieved.
- It brings flexibility to the code.
- It can perform different operations, and hence it eliminates the use of different function names for the same kind of operations.

Disadvantages of Function Overloading in C++

Some of the disadvantages of function overloading are:
1. Functions with different return types cannot be overloaded as they can have the same parameter definition.

Consider the case below:

```
public void num(int a) {
cout << "a = "<<a<<endl;
}

public int num(int a) {
return a + 10;
}
```

In this case, the compiler cannot decide which function to call as both have the same parameter definition even after having different return types.
2. It cannot overload functions with the same name and parameter if any one of them is a static member function declaration.

The static member functions can't be overloaded because the definition must be the same for all class instances. If an overloaded function has many definitions, none of them can be made static

Causes of function overloading in C++

The situation in which the compiler is unable to decide the appropriate overloaded function is called overloading ambiguity. In that case, the compiler won't run the program.
Overloading ambiguity occurs in the following cases:

1. Type conversion

In C++, some data types will get automatically converted to some other data type if the suffix is not mentioned. In that case, the compiler cannot decide which function to call, and an ambiguity error occurs.

C++

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

```
void function(float) {
   cout << "Data Type: float\n";
}

void function(int) {
   cout << "Data Type: int\n";
}

int main() {
  function(1.0);
  function(1);
  return 0;
}
```

In C++, all floating-point constants are considered as double unless explicitly specified by a suffix, so the above code generates a type conversion error. To overcome this problem, we can add a suffix to the passed value.

C++

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

void function(float a) {
   cout << "Data Type: float\n";
}

void function(int a) {
   cout << "Data Type: int\n";
}

int main() {
  // float argument passed
  function(1.0f);
  // int argument passed
  function(1);
  return 0;
}
```

2. Function with default arguments

When a function is overloaded with a default argument, the compiler gets confused if another function satisfies the parameter conditions.
In the example below, when add(a) is called , both add(int a) and add(int a, int b = 10) conditions are fulfilled. In this case, the compiler cannot select which function to call and produces an ambiguity error.
Example:

C++

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

int add(int a) {
  int b = 10;
  return a + b;
}

// function contains a default argument
int add(int a, int b = 10) {
  return a + b;
And }

int main() {
  int a = 5;
  cout << "a + b = "<<add(a)<<endl;
  return 0;
}
```

Error:

3. Function with pass by reference

When a function is overloaded with a reference parameter, the compiler gets confused as there is no syntactical difference between both functions.

C++

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

void display(int a) {
  cout << "a = "<<a<<endl;
}

void display(int &a) {
  cout << "a = "<<a<<endl;
}

int main() {
  int a = 5;
  display(a);
  return 0;
}
```

Error :

There is no syntactical difference between display(a) and display(&a). And in this case, the compiler will not be able to decide which function to call, resulting in an error.

Can the main() function be overloaded in C++?

Yes, the main() function can be overloaded in C++. To overload the main() function, we have to use a class and declare the main() function as a member function.
Example:

C++

```
#include <iostream>
```

```
using namespace std;

// create a Main class and declare main() as member function
class Main {
public:
    int main(int a) {
         cout<<"a = "<<a<<endl;
        return 0;
    }

    int main(int a ,int b) {
        cout<<"a = "<<a<<"; b = "<<b<<endl;
        return 0;
    }
};

int main() {
    Main object;
    object.main(5);
    object.main(5,10);
    return 0;
}
```

Output:

```
a = 5
a = 5; b = 10
```