Constructor with default arguments-

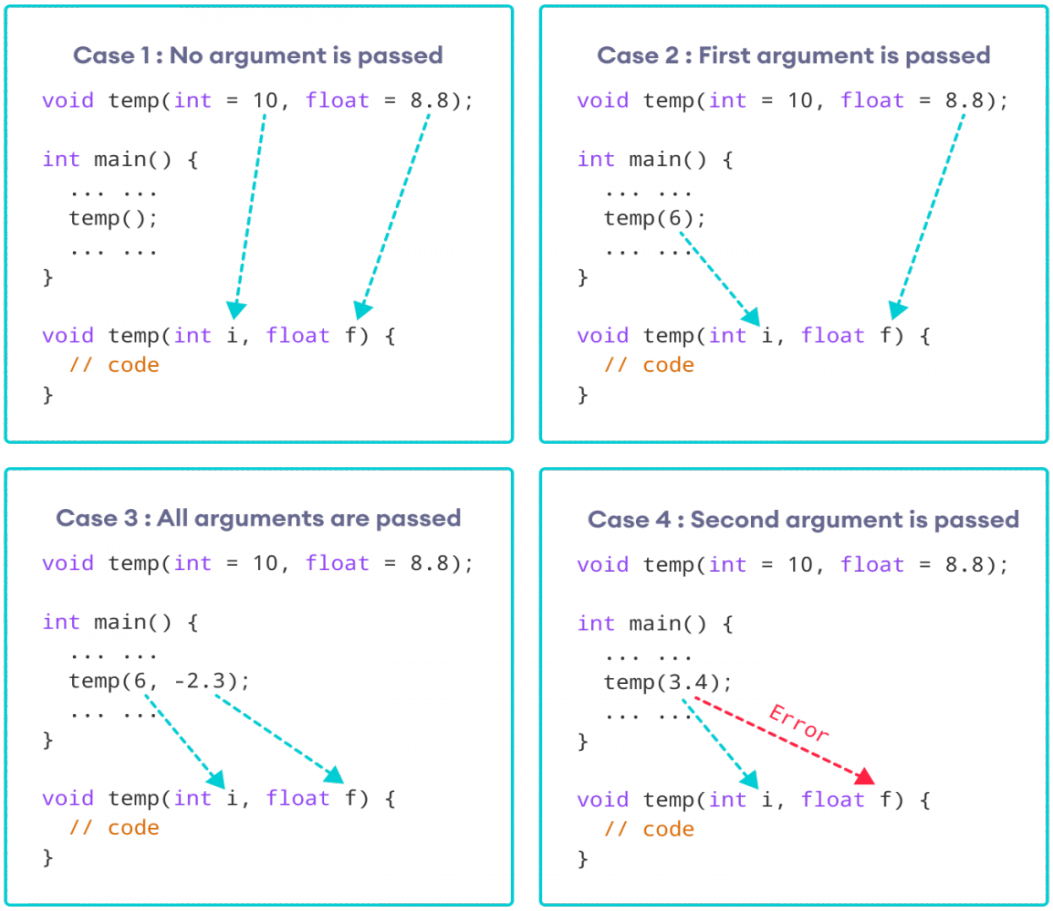
A default argument is a value provided in a function declaration that is automatically assigned by the compiler if the calling function doesn't provide a value for the argument. In case any value is passed, the default value is overridden.

1) The following is a simple C++ example to demonstrate the use of default arguments. Here, we don't have to write 3 sum functions; only one function works by using the default values for 3rd and 4th arguments.

```
// CPP Program to demonstrate Default Arguments
#include <iostream>
using namespace std;

// A function with default arguments,
// it can be called with
// 2 arguments or 3 arguments or 4 arguments.
int sum(int x, int y, int z = 0, int w = 0) //assigning default values to z,w as 0
{
    return (x + y + z + w);
}

// Driver Code
int main()
{
    // Statement 1
    cout << sum(10, 15) << endl;

    // Statement 2
    cout << sum(10, 15, 25) << endl;

    // Statement 3
    cout << sum(10, 15, 25, 30) << endl;
    return 0;
}
```

- **Characteristics for defining the default arguments –**
  - The values passed in the default arguments are not constant. These values can be overwritten if the value is passed to the function. If not, the previously declared value retains.
  - During the calling of function, the values are copied from left to right.

Working of default arguments



- **Destructors –**

Destructor is an instance member function that is invoked automatically whenever an object is going to be destroyed. Meaning, a destructor is the last function that is going to be called before an object is destroyed.

- A destructor is also a special member function like a constructor. Destructor destroys the class objects created by the constructor.
- Destructor has the same name as their class name preceded by a tilde (~) symbol.
- It is not possible to define more than one destructor.
- The destructor is only one way to destroy the object created by the constructor. Hence destructor can-not be overloaded.
- Destructor neither requires any argument nor returns any value.
- It is automatically called when an object goes out of scope.
- Destructor release memory space occupied by the objects created by the constructor.
- In destructor, objects are destroyed in the reverse of an object creation.

The syntax for defining the destructor within the class:

```
~ <class-name>() {
   // some instructions
}
```

The syntax for defining the destructor outside the class:

```
<class-name> :: ~<class-name>() {
   // some instructions
}
```

```
// C++ program to demonstrate the execution of constructor
// and destructor

#include <iostream>
using namespace std;

class Test {
public:
    // User-Defined Constructor
    Test() { cout << "\n Constructor executed"; }

    // User-Defined Destructor
    ~Test() { cout << "\nDestructor executed"; }
};
main()
{
    Test t;

    return 0;
}
```

Output

```
Constructor executed
Destructor executed
```

Example:-

```
// C++ program to demonstrate the number of times
// constructor and destructors are called

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

```
static int Count = 0;        //It is static so that every class object has the same value

class Test {
public:                // User-Defined Constructor
    Test()
    {

            // Number of times constructor is called
            Count++;
            cout << "No. of Object created: " << Count<< endl;
    }

    // User-Defined Destructor
    ~Test()
    {

            cout << "No. of Object destroyed: " << Count << endl;
                                    //It will print count in descending order
            Count--;                // Number of times destructor is called
    }
};

int main()
{
    Test t, t1, t2, t3;

    return 0;
}
```