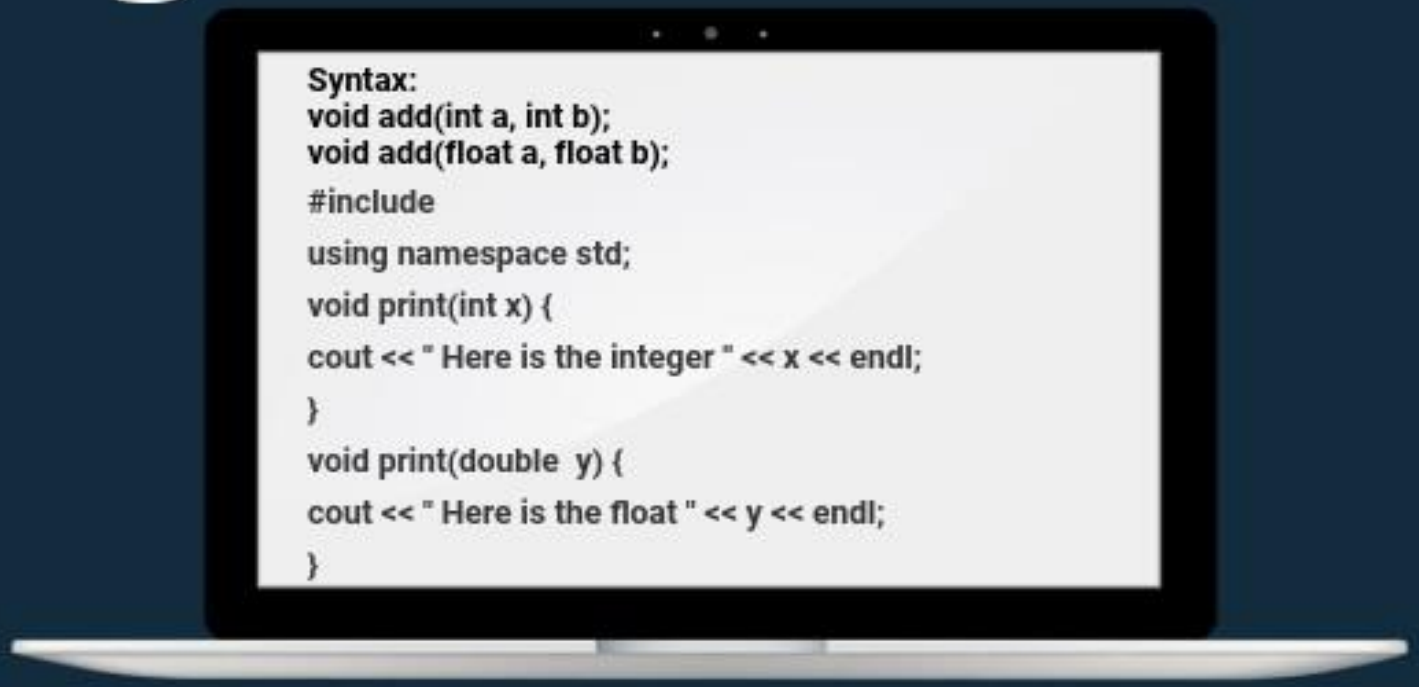
Function Overloading in C++



```
Syntax:
void add(int a, int b);
void add(float a, float b);
#include
using namespace std;
void print(int x) {
cout << " Here is the integer " << x << endl;
}
void print(double  y) {
cout << " Here is the float " << y << endl;
}
```

By – Mohammad Imran

Function Overloading

Introduction

Function overloading is a feature of object-oriented programming where two or more functions can have the same name but different parameters. When a function name is overloaded with different jobs it is called Function Overloading. In Function Overloading “Function” name should be the same and the arguments should be different. Function overloading can be considered as an example of a **polymorphism** feature in C++.

Function Overloading

Introduction

One of the major advantages of Function overloading is that it increases the readability of the program because we don't need to use different names for the same action again and again.

We can overload the function by doing change in –

- ✓ Number of parameter
- ✓ Types of parameter
- ✓ Sequence of parameter

Function Overloading

Number of Parameter

By changing in number of parameter

```
void add();
void add(int);
void add(int,int);
void add(int,int,int);
```

Here add function is overloaded with different number of parameter.

Function Overloading

Types of Parameter

By changing in types of parameter

```
void add();
void add(int);
void add(float);
void add(double);
```

Here add function is overloaded with different types of parameter.

Function Overloading

Sequence of Parameter

By changing in sequence of parameter

```
void add();
void add(int, float);
void add(float, int);
void add(int, float, int);
void add(float, float, float);
```

Here add function is overloaded with different sequence of parameter.

Function Overloading

Advantages

- ✓ We use function overloading to save the memory space, consistency, and readability of our program.
- ✓ With the use function overloading concept, we can develop more than one function with the same name
- ✓ Function overloading shows the behavior of polymorphism that allows us to get different behavior, although there will be some link using the same name of the function.
- ✓ Function overloading speeds up the execution of the program.

Function Overloading

Advantages

- ✓ Function overloading is used for code reusability and also to save memory.
- ✓ It helps application to load the class method based on the type of parameter.
- ✓ Code maintenance is easy.

Function Overloading

Example - 1

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

class FunOverload
{
   public:
   void add(int a, int b)
   {
      cout << "Sum = " << a+b << endl;
   }
   void add(int a, int b, int c)
   {
      cout << "Sum = " << a+b+c << endl;
   }
};
```

```
int main()
{
   FunOverload fo;
   fo.add(10,20);
   fo.add(10,20,30);
   return 0;
}
```

OUTPUT

```
Sum = 30
Sum = 60
```

By – Mohammad Imran

Function Overloading

Example - 2

```
#include <iostream>
using namespace std;
#define PI 3.14
int area(int a)
{
   return a * a;
}
int area(int a, int b)
{
   return a * b;
}
double area(float radius)
{
   return PI * radius * radius;
}
```

By – Mohammad Imran

Function Overloading

Example - 2

```
int main()
{
   int length = 5, breadth = 10;
   float radius = 5.5;
   int area_square = area(length);
   double area_circle = area(radius);
   int area_rectangle = area(length, breadth);

   cout<<"Square area = "<<area_square<<endl;
   cout<<"Rectangle area = "<<area_rectangle<<endl;
   cout<<"Circle area = "<<area_rectangle<<area_circle<<endl;
   return 0;
}
```

OUTPUT

```
Square area = 25
Rectangle area = 50
Circle area = 5094.985
```

By – Mohammad Imran

Call By Value and Call By Reference

Functions can be invoked in two ways: **Call by Value** or **Call by Reference**. These two ways are generally differentiated by the type of values passed to them as parameters.

The parameters passed to function are called ***actual parameters*** whereas the parameters received by function are called ***formal parameters***.

Function

Call By Value and Call By Reference

```
class Class_Name
{
   void function_Name( int a, int b )
   {
      Statements;
   }
};

int main()
{
   Class_Name object;
   Object.function_Name( 100, 200 );
   return 0;
}
```

Formal Parameter

Actual Parameter

By – Mohammad Imran

Function

Call By Value

Call by value is also known as pass by copy or call by value. By definition, pass by value means to pass in the actual parameter's copy in memory as the formal parameters.

Function

Call By Value

Let's understand what happens when we pass a value to a function via pass by value method:

- ✓ We pass in the actual parameters.
- ✓ A copy of the same is created in the memory.
- ✓ This copy is passed as the formal parameters.

Now since we are passing a copy, both the parameters are stored in different memory locations. Thus, any change inside the function will not have any effect on the original values.

Call By Value

When to Use?

- ✓ When we want to perform calculations without changing the original values.
- ✓ To prevent the values from changing via any other thread while multithreading (Multithreading in simple words is running two or more parts of a program concurrently.)

Call By Value

Advantages

- ✓ Functions are free to modify the passed values without affecting the original data.
- ✓ Great while working with multithreaded or asynchronous programs where we need to keep track of the values.
- ✓ Pass by value method can be used to convert a string from an integer without modifying its value.

Call By Value

Disadvantages

- ✓ Since pass-by-value is implemented by passing a copy of the data, the data size increases (doubles). This may not be an issue with smaller programs but can cost efficiency in larger programs.
- ✓ If the argument is too large, copying the values can take a significant amount of time.
- ✓ There is no way to propagate back the updated values through the parameters.

Call By Value

Example - 3

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

void swap(int a, int b);
int main()
{
   int x = 5, y = 15;
   cout << "x = " << x;
   cout << " y = " << y << endl;
   cout << "After swap call" << endl;
   swap(x, y);
   cout << "x = " << x;
   cout << " y = " << y << endl;
   return 0;
}
```

OUTPUT

```
x = 5 y = 15
x = 5 y = 15
```

```
void swap(int a, int b)
{
     int temp;
     temp = a;
     a = b;
     b = temp;
}
```

By – Mohammad Imran

Function Call By Reference

Call by reference is also known as pass-by address or call by reference. While calling a function, instead of passing the values of variables, we pass a reference to values.

Both the actual and formal parameters refer to the same locations, so any changes made inside the function are actually reflected in actual parameters of the caller.

Function Call By Reference

Let's understand what happens when we pass a Reference to a function via pass-by Reference:

- ✓ We pass in the actual parameters.
- ✓ A reference variable is created and passed to the function.
- ✓ A reference variable is a nickname for any variable.

Now since we are passing a copy of the address, both the variables point to the same value. And thus, any change in the function would be reflected in the actual values.

Call By Reference

When to Use?

- ✓ When we are changing the parameters passed by the client program.
- ✓ When passing large structures and classes to make the program efficient.
- ✓ To avoid copying the values and reducing the space complexity.

Call By Reference

Advantages

- ✓ We can produce efficient programs since no copy of the data is made.
- ✓ It enhances the ease of exchanging information between functions through the use of parameters.

Call By Reference

Example - 4

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

void swap(int *a, int *b);
int main()
{
   int x = 5, y = 15;
   cout << "x = " << x;
   cout << " y = " << y << endl;
   cout << "After swap call" << endl;
   swap(&x, &y);
   cout << "x = " << x;
   cout << " y = " << y << endl;
   return 0;
}
```

```
void swap(int *a, int *b)
{
     int temp;
     temp = *a;
     *a = *b;
     *b = temp;
}
```

OUTPUT

```
x = 5 y = 15
x = 5 y = 15
```

By – Mohammad Imran

Call By Value and Call By Reference

Differences

Call By Value	Call By Reference
Also known as pass by copy and call by value.	Also known as pass by address and call by reference.
Copies of the original values are passed to the function.	Only a reference variable (alias name) is passed to the function.
Any change in the function will not be reflected in the original values.	Every change in values is reflected in the actual values.
Values are passed just like it’s done for any program.	Pointer variables are passed to refer to the address location.

By – Mohammad Imran

Return By Reference

Introduction

A reference is represented by the ampersand symbol (**&**) in C++ and is an alias or copy of the original variable. A shallow copy in C++ is made to create a reference variable, which means that the reference variable points to the address of the original variable, and any changes made to the reference variable will be reflected in the original variable. For example, if **b** is a shallow copy of **a**, then if the value of **b** changes, then the value of **a** will also change because **b** will have the same address as **a**.

Changing Variable Value By Reference

Example - 5

```
#include<iostream>
using namespace std;
class Reference
{
  public:
   void display(int a)
   {
      int x = a;
      int &num = x;
      cout << "X = " << x << endl;
      x = x + 5;
      cout << "Num = " << num << endl;
      num = num + 10;
      cout << "X = " << x << endl;
   }
};
```

```
int main()
{
   Reference r;
   r.display(10);
   return 0;
}
```

OUTPUT

```
x = 10
Num = 15
x = 25
```

By – Mohammad Imran

Return By Reference

Introduction

Pointers and References in C++ held close relation with one another. The major difference is that the pointers can be operated on like adding values whereas references are just an **alias** for another variable.

Functions in C++ can return a **reference** as it's returns a **pointer**.

When function returns a **reference** it means it returns a **implicit** pointer.

Return By Reference

Introduction

Return by reference is very different from **Call by reference**. Functions behaves a very important **role** when variable or pointers are returned as reference.

See this function signature of Return by Reference Below:

dataType& functionName(parameters);

Return By Reference

Example - 6

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

int x;
int& retByRef()
{
     return x;
}

int main()
{
     retByRef() = 10;
     cout << << " X = " << x;
     return 0;
}
```

OUTPUT

```
X = 10
```

By – Mohammad Imran

Coding Question

By – Mohammad Imran

Question - 1

Write a program using the C++ programming language to print all the prime numbers between two given numbers by creating a function.

Click here to see code

By – Mohammad Imran

Question - 2

Write a Program to print the Fibonacci sequence of a number using functions in C++ programming language.

Input – 8

Output – 0 1 1 2 3 5 8 13

Click here to see code

By – Mohammad Imran

QUIZ

By – Mohammad Imran

Quiz - 1

Predict the output of given code snippet

```
#include<iostream>
using namespace std;
int abc(int i)
{
     return (i++);
}
int main()
{
     int i = abc(10);
     printf("%d",--i);
}
```

OUTPUT

9

By – Mohammad Imran

Quiz - 2

Predict the output of given code snippet

```
#include <iostream>
using namespace std;
int i = 6;
display()
{
    i = 5;
}
int main()
{
    display();
    cout << i << endl;
    return 0;
}
```

OUTPUT

5

By – Mohammad Imran

Quiz - 3

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

void my_func();
int main()
{
    int my_num = 7;
    cout << my_num;
    my_func();
    return 0;
}
void my_func()
{
   cout << my_num;
}
```

OUTPUT

Error on my_num

By – Mohammad Imran

Quiz - 4

```
#include <iostream>
using namespace std;
int fun(char *str1)
{
    char *str2 = str1;
    while (*++str1);
     return (str1 - str2);
}
int main()
{
    char *str = "Computer";
    cout << fun(str);
    return 0;
}
```

OUTPUT

Error on my_num

By – Mohammad Imran