Introduction to C++

Function Overloading

In This Lecture We Learn

- Function Overloading
- Different Type of Function Overloading
- Examples of Functions Overloading

Function Overloading in C++

Functions with the same name , but different set of parameters can be defined.

Function Overloading in C++

Examples of Function Overloading

```
int test() { }
int test(int a) { }
float test(double a) { }
int test(int a, double b) { }
```

- Here, all 4 functions are overloaded functions because argument(s) passed to these functions are different.
- Notice that, the return type of all these 4 functions are not same. Overloaded functions may or may not have different return type but it should have different argument(s).

Function Overloading in C++

```
// Error in code
int test(int a)
{ }
double test(int b)
{ }
```

- The number and type of arguments passed to these two functions are same even though the return type is different. Hence, the compiler will throw error.

Function Overloading - Rules

- Functions **name** must be **same** e.g. Square, Sum, Cube.
- Changes can be occur in:
- 1) **Return type** may be change
  e.g. **double** Area(), **int** Area() etc.
- 2) **Number** of **arguments** may be change
  e.g. **int** Sum(**int**, **int**), e.g. **int** Sum(**int**, **int**, **int**)
- **Data types** of arguments may be change
  e.g. **int** Square(**int**), **int** Square(**double**)

Why We Use Function Overloading

- **Duplication** of name of functions
- **Save** memory
- Compile time binding
- Batter **readability**

Example - Overloaded Square Functions – 1/2

```
#include <iostream>
using namespace std;
int Square (int number)
{
   return number * number;
}
double Square (double number)
{
   return number * number;
}
```

Example - Overloaded Square Functions – 1/2

```
#include <iostream>
using namespace std;
int Square (int number)
{
    return number * number;
}
double Square (double number)
{
    return number * number;
}
```

Function having Integer type Argument

Example - Overloaded Square Functions – 1/2

```
#include <iostream>
using namespace std;
int Square (int number)
{
    return number * number;
}
double Square (double number)
{
    return number * number;
}
```

Function having double type Argument

Example - Overloaded Square Functions – 1/2

```
#include <iostream>
using namespace std;
int Square (int number)
{
    return number * number;
}
double Square (double number)
{
    return number * number;
}
```

Both Functions Having Same Name

Example - Overloaded Square Functions – 2/2

```
int main()
{
   int iNo = 10;
   double dNo = 10.10;
   cout<<"The square of integer is: "<< Square(iNo)<< endl;
   cout<<"The square of double is:"<<Square(dNo)<< endl;
return 0;
}
```

```
The square of integer is: 100
The square of double is: 102.01
```

Another Example is - 1/2

```
/*Calling overloaded function test() with different arguments.*/
#include <iostream>
using namespace std;
void test(int);
void test(float);
void test(int, float);
int main() {
        int a = 5 ; float b = 5.5;
        test(a) ; test(b) ; test(a, b);
 return 0;
}
```


Another Example is - 2/2

```
void test(int var) {
        cout<<"Integer number: "<<var<<endl;
}

void test(float var){
        cout<<"Float number: "<<var<<endl;
}

void test(int var1, float var2) {
        cout<<"Integer number: "<<var1;
        cout<<" And float number:"<<var2;
}
```


Output of the Previous Program

```
Integer number: 5
Float number: 5.5
Integer number: 5 And float number: 5.5
```

In above example, function test() is called with integer argument at first. Then, function test() is called with floating point argument and finally it is called using two arguments of type int and float. Although the return type of all these functions is same, that is, void, it's not mandatory to have same return type for all overloaded functions.

Function Overloading

- **Different ways to overload the method**
- There are two ways to overload the method in C++
  - By changing number of arguments or parameters
  - By changing the data type

By Changing Number of Arguments

- In this type of function overloading we define two functions with same names but different number of parameters of the same type.
- For example, in the next slide mentioned program we have made two sum() functions to return sum of two and three integers.

By Changing Number of Arguments

```
#include<iostream>
using namespace std;
void sum(int a, int b)
{
        cout<<a+b;
}
void sum(int a, int b, int c)
{
        cout<<a+b+c;
}
int main()
{
 sum(10, 20);
 cout<<endl;
 sum(10, 20, 30);
}
```


By Changing Number of Arguments

```
#include<iostream>
using namespace std;
void sum(int a, int b)
{
        cout<<a+b;
}
void sum(int a, int b, int c)
{
        cout<<a+b+c;
}
int main()
{
 sum(10, 20);
 cout<<endl;
 sum(10, 20, 30);
}
```

Function Having Same Name

```
30
60
```


By Changing Number of Arguments

```
#include<iostream>
using namespace std;
void sum(int a, int b)
{
        cout<<a+b;
}
void sum(int a, int b, int c)
{
        cout<<a+b+c;
}
int main()
{
 sum(10, 20);
 cout<<endl;
 sum(10, 20, 30);
}
```

Function With Two Parameters

By Changing Number of Arguments

```
#include<iostream>
using namespace std;
void sum(int a, int b)
{
        cout<<a+b;
}
void sum(int a, int b, int c)
{
        cout<<a+b+c;
}
int main()
{
 sum(10, 20);
 cout<<endl;
 sum(10, 20, 30);
}
```

Function With Three Parameters

Different Datatype of Arguments

- In this type of overloading we define two or more functions with same name and same number of parameters, but the type of parameter is different.
- For example in this next slide program, we have two sum() function, first one gets two integer arguments and second one gets two double arguments.

Different Datatype of Arguments

```
#include<iostream>
using namespace std;
void sum(int a, int b) {
        cout<<a+b;
}
void sum(double a, double b) {
        cout<<a+b;
}
int main()
{
        sum(10, 20);
        cout<<endl;
        sum(10.5, 24.5);
}
```


Different Datatype of Arguments

```
#include<iostream>
using namespace std;
void sum(int a, int b) {
        cout<<a+b;
}
void sum(double a, double b) {
        cout<<a+b;
}
int main()
{
        sum(10, 20);
        cout<<endl;
        sum(10.5, 24.5);
}
```

Function Parameters Having Integer Data Type

Different Datatype of Arguments

```
#include<iostream>
using namespace std;
void sum(int a, int b) {
        cout<<a+b;
}
void sum(double a, double b) {
        cout<<a+b;
}
int main()
{
        sum(10, 20);
        cout<<endl;
        sum(10.5, 24.5);
}
```

Function Parameters Having Float Data Type

Different Datatype of Arguments

```
#include<iostream>
using namespace std;
void sum(int a, int b) {
        cout<<a+b;
}
void sum(double a, double b) {
        cout<<a+b;
}
int main()
{
        sum(10, 20);
        cout<<endl;
        sum(10.5, 24.5);
}
```

Both Functions Having Same Number of Parameters

```
30
35
```


Function Overloading

```
int absolute(int);
float absolute(float);
int main() {
   int a = -5;   float b = 5.5;
   cout << "Absolute value of " << a << " = " << absolute(a) << endl;
   cout << "Absolute value of " << b << " = " << absolute(b);
   return 0;
}
int absolute(int var) {
    if (var < 0)
       var = -var;
   return var;
}
float absolute(float var){
   if (var < 0.0)
      var = -var;
   return var;
}
```

```
Absolute value of -5 = 5
Absolute value of 5.5 = 5.5
```


Explanation of Previous Program

- In the previous example, two functions absolute() are overloaded.
- Both functions take single argument. However, one function takes integer as an argument and other takes float as an argument.
- When absolute() function is called with integer as an argument, this function is called:

```
int absolute(int var) {
   if (var < 0)
      var = -var;
   return var;
}
```

- When absolute() function is called with float as an argument, this function is called:

```
float absolute(float var){
   if (var < 0.0)
      var = -var;
   return var;
```

Function Overloading

- **How function calls are matched with overloaded functions**
- Making a call to an overloaded function results in one of three possible outcomes:
  - 1) A **match is found**. The call is resolved to a particular overloaded function.
  - 2) **No match is found**. The arguments can not be matched to any overloaded function.
  - 3) **An ambiguous match** is found. The arguments matched more than one overloaded function.

Function Overloading

- **How function calls are matched with overloaded functions**
- When an overloaded function is called, C++ goes through the following process to determine which version of the function will be called:
  - 1-First, C++ tries to find an exact match. This is the case where the actual argument exactly matches the parameter type of one of the overloaded functions. For example:

```
void print(char *value);
void print(int value);

print(0); // exact match with print(int)
```


Function Overloading

- **How function calls are matched with overloaded functions**
- When an overloaded function is called, C++ goes through the following process to determine which version of the function will be called:
  - 1-First, C++ tries to find an exact match. This is the case where the actual argument exactly matches the parameter type of one of the overloaded functions. For example:

Although 0 could technically match print(char*) (as a null pointer), it exactly matches print(int). Thus print(int) is the best match available

```
void print(char *value);
void print(int value);

print(0); // exact match with print(int)
```


Function Overloading

- **How function calls are matched with overloaded functions**
  - 2-If no exact match is found, C++ tries to find a match through promotion. Certain types can be automatically promoted via internal type conversion to other types.
  - i.e. Char, unsigned char, and short is promoted to an int.

```
void print(char *value);
void print(int value);

print('a'); // promoted to match print(int)
```

  - In this case, because there is no print(char), the char 'a' is promoted to an integer, which then matches print(int).

Function Overloading

- **How function calls are matched with overloaded functions**
  - 3-If no promotion is found, C++ tries to find a match through standard conversion. Standard conversions include:
    - Any numeric type will match any other numeric type, including unsigned (eg. int to float)
    - Zero will match a pointer type and numeric type (eg. 0 to char*, or 0 to float)
    - A pointer will match a void pointer

Thank You ☺