Function

Overloading function

Function overloading is a feature in C++ where two or more functions can have the same name but different parameters.

For Example: write three functions with same name and different with parameter (int, double, and float). The function take a value then apply the multiplication operation by number 2.

```
#include <iostream>

using namespace std;

void mul(int i)

{

  cout << "Execute function int: " <<i << " * 2 = "<<  i*i<< endl;

}

void mul(double d)

{

  cout << "Execute function double: " <<d << " * 2 = "<< d*d << endl;

}

void mul(float f)

{

  cout << "Execute function float: " <<f << " * 2 = "<<f*f << endl;

}


int main ()

{

  mul(2);

  mul(4.5);

  mul(5482.54);
```

}

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

void mul(int i)
{
    cout << "Execute function int: " <<i << " * 2 = "<<  i*i<< endl;
}
void mul(double d)
{
    cout << "Execute function double: " <<d << " * 2 = "<< d*d << endl;
}
void mul(float f)
{
    cout << "Execute function float: " <<f << " * 2 = "<<f*f << endl;
}

int main ()
{
    mul(2);
    mul(4.5);
    mul(5482.54);

}
```

```
Select F:\BST\C++-language\C++-codes\Function2\bin\Debug\Function2.exe
Execute function int: 2 * 2 = 4
Execute function double: 4.5 * 2 = 20.25
Execute function double: 5482.54 * 2 = 3.00582e+007

Process returned 0 (0x0)   execution time : 0.266 s
Press any key to continue.
```

Exercise: write two functions with same name sum. The function take two value then apply the summation operation then return result.

#include <iostream>

using namespace std;

int sum (int a, int b)

{

 return a+b;

}

double sum (double a, double b)

{

 return a+b;

}

int main ()

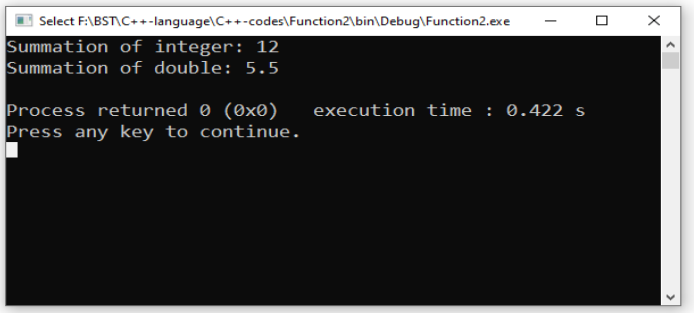
{

cout<<"Summation of integer: "<<sum(4,8)<<endl;

cout<<"Summation of double: "<<sum(2.5,3.0)<<endl;

}

```
1  #include <iostream>
2  using namespace std;
3
4  int sum (int a, int b)
5  {
6     return a+b;
7  }
8
9  double sum (double a, double b)
10 {
11    return a+b;
12 }
13
14
15 int main ()
16 {
17    cout<<"Summation of integer: "<<sum(4,8)<<endl;
18    cout<<"Summation of double: "<<sum(2.5,3.0)<<endl;
19
20 }
```



Types of function call

- **Call by value**: this method copies the actual value of an argument into the formal parameter of the function. In this case, changes made to the parameter inside the function have no effect on the argument.
- **Call by reference**: this method copies the reference of an argument into the formal parameter. Inside the function, the reference is used to access the actual argument used in the call.
- **Call by pointer**: this method copies the address of an argument into the formal parameter. Inside the function, the address is used to access the actual argument used in the call. This means that changes made to the parameter affect the argument.

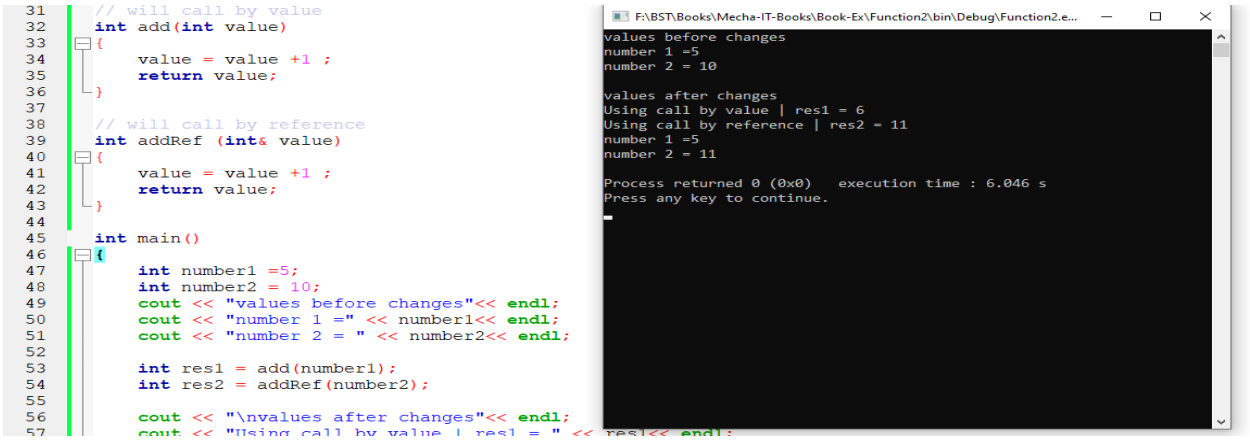
Call by value VS Call by reference

Note: The operator & used with function parameter that make function call by reference.

For Example: I will create a two function that increase a value of number by 1. The first function can't change the actual value (Call by value) in the main function while the second function change the actual value (Call by reference) in the main function.

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

// will call by value
int add(int value)
{
    value = value +1 ;
    return value;
}

// will call by reference
int addRef (int& value)
{
    value = value +1 ;
    return value;
}

int main()
{
    int number1 =5;
    int number2 = 10;

    cout << "values before changes"<< endl;
    cout << "number 1 =" << number1<< endl;
    cout << "number 2 = " << number2<< endl;
    int res1 = add(number1);
    int res2 = addRef(number2);
    cout << "\nvalues after changes"<< endl;
```

```
    cout << "Using call by value | res1 = " << res1<< endl;
    cout << "Using call by reference | res2 = " << res2<< endl;
    cout << "number 1 =" << number1<< endl;
    cout << "number 2 = " << number2<< endl;
    return 0;
}
```



```
// will call by value
int add(int value)
{
    value = value +1 ;
    return value;
}

// will call by reference
int addRef (int& value)
{
    value = value +1 ;
    return value;
}

int main()
{
    int number1 =5;
    int number2 = 10;
    cout << "values before changes"<< endl;
    cout << "number 1 =" << number1<< endl;
    cout << "number 2 = " << number2<< endl;

    int res1 = add(number1);
    int res2 = addRef(number2);

    cout << "\nvalues after changes"<< endl;
```

```
F:\BST\Books\Mecha-IT-Books\Book-Ex\Function2\bin\Debug\Function2.e...
values before changes
number 1 =5
number 2 = 10

values after changes
Using call by value | res1 = 6
Using call by reference | res2 = 11
number 1 =5
number 2 = 11

Process returned 0 (0x0)   execution time : 6.046 s
Press any key to continue.
```

The two function can do the same job.

int addRef (int& value)
{
value = value +1 ;
return value;
}
	void addRef (int& value)
	{
	value ++;
	}

Solution 2:

```
#include <iostream>
using namespace std;
// will call by value
int add(int value)
{
    value = value +1 ;
```

```
    return value;
}
// will call by reference
void addRef2 (int& value)
{
    value ++;
}
int main()
{
    int number1 =5;
    int number2 = 10;
    cout << "values before changes"<< endl;
    cout << "number 1 =" << number1<< endl;
    cout << "number 2 = " << number2<< endl;
    int res1 = add(number1);
    addRef2(number2);
    cout << "\nvalues after changes"<< endl;
    cout << "Using call by value | res1 = " << res1<< endl;
    cout << "number 1 =" << number1<< endl;
    cout << "number 2 = " << number2<< endl;
    return 0;
}
```

```
44  void addRef2 (int& value)
45  {
46      value ++;
47  }
48
49  int main()
50  {
51      int number1 =5;
52      int number2 = 10;
53      cout << "values before changes"<< endl;
54      cout << "number 1 =" << number1<< endl;
55      cout << "number 2 = " << number2<< endl;
56
57      int res1 = add(number1);
58      //int res2 = addRef(number2);
59      addRef2(number2);
60      cout << "\nvalues after changes"<< endl;
61      cout << "Using call by value | res1 = " << res1<<
62      //cout << "Using call by reference | res2 = " <<
63      cout << "number 1 =" << number1<< endl;
64      cout << "number 2 = " << number2<< endl;
65
66
67      return 0;
68  }
69
```

```
F:\BST\Books\Mecha-IT-Books\Book-Ex\Function2\bin\Debug\Func...
values before changes
number 1 =5
number 2 = 10

values after changes
Using call by value | res1 = 6
number 1 =5
number 2 = 11

Process returned 0 (0x0)   execution time : 0.250 s
Press any key to continue.
```

Swap programs:

```
1   #include <iostream>
2   using namespace std;
3
4   int main()
5   {
6       int n1 = 4;
7       int n2= 20;
8       cout << "Before swapping" << endl;
9       cout << "n1 = " << n1 << endl;
10      cout << "n2 = " << n2 << endl;
11
12      int temp;
13      temp = n1;
14      n1 = n2;
15      n2 = temp;
16
17      cout << "\nAfter swapping" << endl;
18      cout << "n1 = " << n1 << endl;
19      cout << "n2 = " << n2 << endl;
20  }
```

```
F:\BST\Books\Mecha-IT-Books\Book-Ex\Function2\bin\De...
Before swapping
n1 = 4
n2 = 20

After swapping
n1 = 20
n2 = 4

Process returned 0 (0x0)   execution time : 0.039 s
Press any key to continue.
```

Exercise: Write a function that swap two numbers without using array.

Notes: The function return only one value so you can't return the two numbers you changed. I will use a reference operator to change the values directly without used return.

```
3
4   void swap(int& n1, int& n2)
5   {
6       int temp;
7       temp = n1;
8       n1 = n2;
9       n2 = temp;
10  }
11  int main()
12  {
13      int n1 = 4;
14      int n2= 20;
15      cout << "Before swapping" << endl;
16      cout << "n1 = " << n1 << endl;
17      cout << "n2 = " << n2 << endl;
18
19      /*int temp;
20      temp = n1;
21      n1 = n2;
22      n2 = temp;*/
23
24      swap(n1,n2);
25
26      cout << "\nAfter swapping" << endl;
27      cout << "n1 = " << n1 << endl;
28      cout << "n2 = " << n2 << endl;
```