Introduction To Templates

This lesson introduces the concept of function templates in C++

We'll cover the following

Definition

Templates are the mechanism by which C++ implements the generic concept.

The following example illustrates **two** *non-generic* (type-sensitive) functions for **multiplying** two numbers, `x` and `y`:

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

int multiply ( int x, int y ) //multiplies two ints
{
    return (x * y);
}

double multiply ( double x, double y )  //multiplies two doubles
{
    return (x * y);
}

int main(){
  int temp1;
  double temp2;
  temp1 = multiply(4,5);
  temp2 = multiply(4.5,5.5);
  cout << "Value of temp1 is: "<< temp1<<endl;
  cout << "Value of temp2 is: "<<temp2<<endl;
}
```

Two *functions* that do exactly the same thing, but cannot be defined as a single function because they use *different* data types.

Function Templates

Templates were made to fulfill the need to design a **generic** code, that works the same way in different situations.

Syntax

To start a *template*, you must provide the following *declaration*:

```
template<class Type>
// or
template<typename Type>
```

The keywords `class` and `typename` have exactly the same meaning in this case, but some compilers may replace the words with each other.

`Type` is our *generic* data type's name, and when the template is to be used, it would be the same as if `Type` was a `typedef` for your datatype,

The following example now illustrates how the `multiply` *function* would be written using a *template*:

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


template<class Type>
Type multiply ( Type x, Type y )
{
    return (x * y);
}

int main() {
  int result = multiply<int> ( 2, 5 ); //calling template type function
  cout << "Result when integer values are passed is: " << result <<endl;
  double result2 = multiply<double>(2.5,5.5);
  cout << "Result when double values are passed is: "<<result2;
  return 0;
}
```

> Note: Optionally, a template can have more type options, and the syntax is pretty simple.

For a template with **three** types, called `First`, `Second` and `Third`, we have:

For a template with **three** types, called `First`, `Second` and `Third`, we have:

```
template<class First, class Second, class Third>
```

In the next lesson, we'll take a look at templates in some more detail!