**Non-type template arguments.**

The template parameters can not only include types introduced by *class* or *typename*, but can also include expressions of a particular type:

|  |  |  |  |
| --- | --- | --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 | // template arguments  #include <iostream>  using namespace std;  template <class T, int N>  T fixed\_multiply (T val)  {  return val \* N;  }  int main() {  std::cout << fixed\_multiply<int,2>(10) << '\n';  std::cout << fixed\_multiply<int,3>(10) << '\n';  } | 20  30 | [Edit & Run](https://cplusplus.com/doc/tutorial/functions2/) |

The second argument of the *fixed\_multiply* function template is of type *int*. It just looks like a regular function parameter, and can actually be used just like one.

But there exists a major difference: the value of template parameters is determined on compile-time to generate a different instantiation of the function *fixed\_multiply*, and thus the value of that argument is never passed during runtime: The two calls to *fixed\_multiply* in *main* essentially call two versions of the function: one that always multiplies by two, and one that always multiplies by three. For that same reason, the second template argument needs to be a constant expression (it cannot be passed a variable).

Elucidations with instances:

Generally, a C++ template, with a single argument looks like this:

*template<typename template\_name>*

But it has been seen that a template can have multiple arguments. The syntax for the same would be:

*template<class T1, class T2, class T3, ………, class Tn>*

or:

*template< typename T1, typename T2, typename T3, ………, typename Tn>*