13.4 — Template non-type parameters

BY ALEX ON JUNE 19TH, 2008 | LAST MODIFIED BY ALEX ON JANUARY 23RD, 2020

In previous lessons, you've learned how to use template type parameters to create functions and classes that are type independent. However, template type parameters are not the only type of template parameters available. Template classes and functions can make use of another kind of template parameter known as a non-type parameter.

Non-type parameters

A template non-type parameter is a special type of parameter that does not substitute for a type, but is instead replaced by a value. A non-type parameter can be any of the following:

- A value that has an integral type or enumeration
- A pointer or reference to a class object
- A pointer or reference to a function
- A pointer or reference to a class member function
- std::nullptr_t

In the following example, we create a non-dynamic (static) array class that uses both a type parameter and a non-type parameter. The type parameter controls the data type of the static array, and the non-type parameter controls how large the static array is.

```
#include <iostream>

template <class T, int size> // size is the non-type parameter
class StaticArray
{
private:
    // The non-type parameter controls the size of the array
    T m_array[size];

public:
    T* getArray();

    T& operator[](int index)
    {
        return m_array[index];
    }
};

// Showing how a function for a class with a non-type parameter is defined outside of the cla
ss
template <class T, int size>
T* StaticArray<T, size>::getArray()
{
    return m_array;
}

int main()
{
    // declare an integer array with room for 12 integers
    StaticArray<int, 12> intArray;

    // Fill it up in order, then print it backwards
    for (int count=0; count < 12; ++count)
        intArray[count] = count;
```

```
35
36      for (int count=11; count >= 0; --count)
37          std::cout << intArray[count] << " ";
38      std::cout << '\n';
39
40      // declare a double buffer with room for 4 doubles
41      StaticArray<double, 4> doubleArray;
42
43      for (int count=0; count < 4; ++count)
44          doubleArray[count] = 4.4 + 0.1*count;
45
46      for (int count=0; count < 4; ++count)
47          std::cout << doubleArray[count] << ' ';
48
49      return 0;
}
```

This code produces the following:

```
11 10 9 8 7 6 5 4 3 2 1 0
4.4 4.5 4.6 4.7
```

One noteworthy thing about the above example is that we do not have to dynamically allocate the m_array member variable! This is because for any given instance of the StaticArray class, size is actually constant. For example, if you instantiate a StaticArray<int, 12>, the compiler replaces size with 12. Thus m_array is of type int[12], which can be allocated statically.

This functionality is used by the standard library class std::array. When you allocate a std::array<int, 5>, the int is a type parameter, and the 5 is a non-type parameter!

 **13.5 -- Function template specialization**

 **Index**

 **13.3 -- Template classes**

C++ TUTORIAL C++, EXPRESSION PARAMETERS, PROGRAMMING, TEMPLATE SPECIALIZATION, TEMPLATES, TUTORIAL | PRINT THIS POST

50 comments to 13.4 — Template non-type parameters

CHERIS
August 6, 2019 at 9:28 am · Reply

What is difference between typename and class in tamplate.

nascardriver