An Idiot's Guide to C++ Templates - Part 1

Alex Dathskovsky
Feb 2, 2015 at 2:00 AM
in C++ Code Snippets

Copying this page violates your terms of service.

Prologue

Welcome to the first tutorial of the series dealing with C++ templates. Here in this first tutorial we will cover the following topics:

- Function Templates
- Function Template Specialization
- Class Templates
- Class Template Specialization
- Partial Template Specialization
- Template Parameters
- Default Template Arguments
- Function Template Overloading
- Class Template Inheritance
- Class Template Composition
- Class Template Friendship
- Class Template Friend Function
- Class Template Friend Class
- Class Template Friend Template Class
- Class Template Friend Template Function
- Class Template Friend Template Friend Function
- Class Template Friend Template Friend Class
- Class Template Friend Template Friend Template Class
- Class Template Friend Template Friend Template Friend Function

The Syntax Charma

Templates are a powerful tool, but they are very complex. Let's begin slowly. The keyword is `template`. Then, in angle brackets (`<>`), a parameter list. Templates, they are always used together in the form:

```
template <>
class/struct/union/function/variable/alias
```

Following the keyword `template`, the angle brackets (`<>`) are the parameters. Parameters are separated by commas. Parameters can be:

- a type, for example `T`
- a constant expression, for example `N`
- a template, for example `Foo`

Following the keyword `class`, `struct`, `union`, `function`, `variable`, `alias` is the name of the template. Parameters are used in the following way:

- `class` and `struct` are used for class templates
- `union` is used for union templates
- `function` is used for function templates
- `variable` is used for variable templates
- `alias` is used for alias templates

Let's begin with a simple example:

```
template <typename T>
class Foo
{
    T* ptr;
    int* const* const* const
```