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

Alex Sherbakov
Posted by Alex on 07/01/2004 07:00:00
Alex Sherbakov 07/01/2004

Copyright Alex Sherbakov, 2004 - 2011

Prolusion

Welcome to "An Idiot's Guide to C++ Templates, Part 1". Templates are a fairly complicated subject. This course uses a gentle approach:

- Introduce a concept
- Complete simple examples, and very long
- Explain the concept

Each lesson builds upon the previous one, and is carefully constructed, and unique. Nevertheless, the whole thing can be quite long, requiring complete concentration. Many of you have been given homework assignments by your instructors. Many of you have been given a deadline to finish a project. Some of you are in a rush to get a job done. Some of you are just curious. Regardless of your circumstances, I hope that this course will be of benefit to you. I hope that this course will be a pleasant experience for you. I hope that this course will be a rewarding experience for you.

Table of Contents

- The Cast
- The Plot Thickens
- Function Templates
  - Without Templates and Using Function Templates
  - Multiple Types with Function Templates
  - Function Template Parameters
  - Function Template Arguments
  - Function Template Instantiation
  - Function Template Specialization
- Class Templates
  - Class Template Syntax
  - Class Template Arguments
  - Class Template Instantiation
  - Class Template Specialization
  - Class Template Partial Specialization
- Class Template Members
  - Member Function Templates
  - Member Function Template Specialization
  - Member Function Template Partial Specialization

The Cast

Initially, there are only two characters in this story: a template, and a function that uses the template. These are a good simple beginning.

```
template <typename T>
T const & max(T const & a, T const & b);

int const & max(int const & a, int const & b);

int const i = max(1, 2);
int const j = max(3, 4);
int const k = max(5, 6);
```

Initially, the compiler sees the template, and stores it. Later, the compiler sees the function call, and uses the template to create the function. The compiler then sees the function, and stores it. Later, the compiler sees the function call, and uses the function. Later, the compiler sees the function call, and uses the function.

Although the compiler sees the template only once, the compiler uses the template three times. Each time, the compiler uses the template, the compiler creates a function. The compiler creates three functions, one for each call. The compiler creates three functions, one for each call. The compiler creates three functions, one for each call