3 August 2017

Toronto trip report by Andrew Pardoe (MSFT)

Post

- Biggest news: Concepts TS merged into C++ draft standard
  - EWG is committed to bringing back an abbreviated syntax in future meetings, ideally before C++20 is finished. P0694R0 is a paper by Bjarne Stroustrup on the natural syntax.
- Modules TS discussions in detail
- Other proposals described in detail
- Discouraged proposals

Toronto trip report by Botond Ballo

- Post
- Reddit thread
- Detailed EWG report
- Rejected proposals
- Detailed report on Concepts TS

  AFTs have been controversial since their introduction, due to their ability to make template code look like non-template code. Many have argued that this is a bad idea, beacuse template code is fundamentally different from non-template code (e.g. consider different name lookup rules, the need for syntactic disambiguators like typename, and the ability to define a function out of line). Others have argued that making generic programming (programming with templates) look more like regular programming is a good thing.

Botond Ballo's report (cont.)

I really don't understand this desire for a syntactic marker. IDEs are perfectly capable of semantic highlighting #

I don't think it's really so important to recognize templates that they need to be called out that way at all #

more than 30 years ago, when overloading was introduced in C++ there was a requirement of an overload declaration with the overload keyword. It didn't work out #

I actually can't remember if I have ever wanted to know whether a function is a template or not <...> You should not program in 2017 using 20 year old tools, nor should a language feature implemented in 2017 be designed with 20 year old tools in mind. #

<...> not everyone uses editors with semantic highlighting capabilities, and that people often look at code in non-editor contexts like code review tools > The language should not accumulate cruft to accommodate 10 year old tools #

What should the C++ Standards Committee be doing?

1. Compile-time stability: Every change in behavior in a new version of the standard is detectable by a compiler for the previous version.
2. Link-time stability: ABI breakage is avoided except in very rare cases, which will be welldocumented and supported by a written rationale.
3. Compiler performance stability: Changes will not imply significant added compile-time costs for existing code.
4. Run-time Performance stability: Changes will not imply added run-time costs to existing code.
5. Progress: Every revision of the standard will offer improved support for some significant programming activity or community.
6. Simplicity: Every revision of the standard will offer some simplification of some significant programming activity.
7. Timeliness: The next revision of the standard will be shipped on time according to a published schedule.

What should the ISO C++ Standards Committee be doing? (cont.)

- Reddit thread
  - Robert Ramey's comment
  - Z01dbrg's troll comment and responses by Louis Dionne
  - Gabriel Dos Reis's comment

Precompiled header issues and recommendations

Andrew Pardoe et al., Microsoft

- Move to a newer compiler, use x64 compiler
- Multi-CPU systems: Failure to automatically increase the pagefile size (Windows bug)
- Pass `/p:PreferredToolArchtecture=x64` to MSBuild
- Use `/MP` compiler option (Details)
- Don't use `#pragma hdrstop`, use `/Fp<PCH-file-name>` instead

Undocumented MSVC options for build timing

- Compiler: `/Bt+`
- Linker: `/time+`
- Source: Going Native 35

Using Trompeloeil (Björn Fahller)

A mocking framework for modern C++ (C++14, single header)

- NDC Oslo, June 2017
- ACCU 2017, May 2017
- Sweden C++, September 2016
- GitHub (Boost licence)
- Cheatsheet (PDF)

cppcoro: a library of coroutine abstractions

The 'cppcoro' library provides a set of general-purpose primitives for making use of the coroutines TS proposal described in N4628.

GitHub

- Coroutine types: `task<T>`, `lazy_task<T>`, `shared_task<T>`, `shared_lazy_task<T>`, `generator<T>`, `recursive_generator<T>`, `async_generator<T>`
- Awaitable types: `single_consumer_event`, `async_mutex`, `async_manual_reset_event`, `async_auto_reset_event`
- Cancellation: `cancellation_token`, `cancellation_source`, `cancellation_registration`
- Schedulers and I/O: `io_service`, `io_work_scope`, `file`, `readable_file`, `writable_file`, `read_only_file`, `write_only_file`, `read_write_file`