C/C++ LANGUAGE SERVERS
THE NEXT GENERATION IS NOW
AGENDA

- Why C++
- Modern C++
- CDT’s language services
- Clangd and the clang based language servers
- LSP for CDT
- Going forward
WHERE IS C++ TODAY

• From ISO C++: C++ Developer Survey “Lite”: 2018-02
Q4 What types of projects do you work on? (select all that apply)

- Business (e.g., ERP...)
- Communications (e.g.,...)
- Consumer (e.g., retail...)
- Developer tools (e.g.,...)
- Engineering (e.g.,...)
- Entertainment (e.g., sport...)
- Financial (e.g., trad...)
- Frameworks (e.g., React...)
- Gaming (e.g., console and...)
- Hardware/IoT (e.g., embed...)
- Productivity (e.g., budge...)
- Social and business...
- Utility apps (e.g.,...)
- Other (please specify)

Answered: 3,269  Skipped: 17
Q5 What platforms do you develop for? (select all that apply)

Answered: 3,271  Skipped: 15

- Linux (desktop)
- Windows (desktop)
- Server / cloud
- Embedded system
- Mac
- Android
- iOS
- Game console
- Other (please specify)
Q13 Which development environments (IDEs) or editors do you use for C++ development?

Answered: 3,240   Skipped: 46

- Visual Studio
- Vim
- Visual Studio Code
- Qt Creator
- CLion
- Emacs
- Eclipse
- Sublime
- XCode
- Other (please specify)
- Android Studio
Q8 What version(s) of C++ are you allowed to use on your current project (work or school)?

Answered: 3,257  Skipped: 29

- C++11 (e.g., auto, move...)
- C++14 (e.g., generic...)
- C++17 (e.g., if constexpr...)

[Bar chart showing percentages of responses for each version of C++ allowed on current projects.]
MODERN C++ - C++11, C++14, C++17, C++20, ...

- Safety through stack based scoping instead of direct heap
- Ownership and move semantics
- Smart pointers instead of raw pointers
- std::string instead of raw character arrays
- STL containers – vector, list and map
- STL algorithms for generic collection search and manipulation
- Lambdas instead of small functions
  - myfunc([capture_x](param_y) { ... })
- Range-based for loops
  - for(auto &x : myvector) { ... }

QNX
A Subsidiary of BlackBerry
C++ CORE GUIDELINES

- Edited by Bjarne Stroustrup and Herb Sutter
- “The aim is to help C++ programmers to write simpler, more efficient, more maintainable code.”
- A large collection of rules on how to use C++ properly
- Also provide instruction to tool makers on enforcement of the rules
R.1: RESOURCE MANAGEMENT

• Bad

```cpp
void send(X* x, cstring_span destination)
{
    auto port = open_port(destination);
    my_mutex.lock();
    // ...
    send(port, x);
    // ...
    my_mutex.unlock();
    close_port(port);
    delete x;
}
```
R.1: RESOURCE MANAGEMENT

• Good

```cpp
void send(unique_ptr<X> x, cstring_span destination) // x owns the X
{
    Port port{destination}; // port owns the PortHandle
    lock_guard<mutex> guard{my_mutex}; // guard owns the lock
    // ...
    send(port, x);
    // ...
} // automatically unlocks my_mutex and deletes the pointer in x
```
R.1: RESOURCE MANAGEMENT

- What is Port?

class Port {
    PortHandle port;

public:
    Port(cstring_span destination) : port{open_port(destination)} { }
    ~Port() { close_port(port); }
    operator PortHandle() { return port; }

    // port handles can't usually be cloned
    // so disable copying and assignment if necessary
    Port(const Port&) = delete;
    Port& operator=(const Port&) = delete;
};
INTERESTING USES OF C++
MICROCONTROLLER PROGRAMMING

• Popularized with Arduino
• Very resource constrained boards
  – E.g. Uno 32K flash, 2K SRAM, 1K EEPROM
  – Though newer ARM Cortex M0/M4, ESP32 getting more powerful
• I/O through traditional GPIO, I2C, SPI, UART (including over USB)
  – Built to interact with sensors and indicators
• Arduino Sketch (ino) language a slight subset of C++
  – Forward declaration, #include <Arduino.h>, only things missing
• Some boards have more advanced SDKs
  – Like ESP32’s ESP-IDF which includes things like MQTT (AWS IoT)
WEBASSEMBLY (WASM)

- Stack based virtual instruction set
  - Compiled to native at load time
- Operates in a sandbox
  - Memory is an ArrayBuffer also accessible from host
  - Wasm code has no access to host objects
  - Host imports functions to wasm, wasm exports functions to host
  - Only 4 data types: int64, int32, float64, float32, i.e. number
- WebAssembly becoming first class citizen in upcoming clang 8.0
- Emscripten provides a rich emulation environment for C++ code
  - Commonly used for games, e.g. Unity
LANGUAGE SERVERS

- CDT
- Clangd (and a bit on cquery and ccls)
CDT LANGUAGE SERVICES – IN THE BEGINNING

• Standard Eclipse editor services like syntax highlighting, bracket matching
• Search provided using a ctags based index
  – Really only properly supported C
• Then we built our own parsers for C and C++
  – Recursive descent parser with backtracking to deal with ambiguities
  – Great error handling to deal with partial files (like at content assist time)
• Cloned the JDT index to persist symbols
  – While accurate, full indexing didn’t scale well
CDT FAST INDEXER – LIVING UP TO IT’S NAME

• Be faster by doing less, cutting out duplication
  – Parse each header file once
  – Store symbol information in a fast database
  – plug objects from that database into the AST (resolveBindings)
  – Rest of the code that uses the AST just works (mostly)
• Indexing of large projects from over an hour to a few minutes
• Incremental indexing almost unnoticeable
• Accuracy fades with bad code patterns
  – But luckily there aren’t many of those
CDT SCANNER DISCOVERY

• In order to parse properly need to consider the compiler environment
  – Built-in macros, include path of particular compiler
  – Arguments used when compiling each source file
• CDT traditionally scanned built output to find which compiler and it’s args
• GCC, for example, has magic options to give macros and the include path
  – gcc -v -E -P -dD
• CDT’s new Core build can also ask builders for this information
  – CMake provides compile_commands.json
  – Qt’s qmake supports query options
CDT REFACTORYING

- Leverage the LTK started by JDT for managing refactoring
- Ask the AST to produce TextEdit’s needed to affect a change
- Uses Preview UI to see changes before committing
- Standard rename refactoring, including in place
- Other built in structural refactorings
  - E.g. extract constant, extract local variable, extract function
- Fully extendible for custom refactorings
CDT CODE ANALYSIS (CODAN)

• Static analysis of code using the AST
  – Find problems before build and launch
• Configurable to run at different times of the project lifecycle
  – From as you type to only on full builds
  – Some checkers take too long to run all the time
• Numerous built in checkers
  – Coding style, e.g. naming conventions
  – Potential code problems, e.g. assignment in condition, missing return
  – Real semantic errors, e.g. instantiating abstract class
• Fully extensible for other team standards, etc.
OTHER CDT LANGUAGE SERVICE FEATURES

- Code formatter
- Call hierarchy
- Type hierarchy
- Include hierarchy
- Organize/Add Includes
- Explore Macro Expansion
CDT’S BIGGEST CHALLENGE

- C++ standards being released every three years
  - With pretty major language changes each time
  - Compilers will often adopt the features before the standard
- How do we keep up?
  - All time spent on updating parsers and index
  - No time for new refactorings, code analysis checks
  - And the community is now very small
LANGUAGE SERVER PROTOCOL

- Provides language smarts for editors
- Code completion
- Go to declaration, find references
- Workspace symbols (search), document symbols (outline)
- Hover
- Diagnostics and Code actions (quick fix)
- Code formatting
- Rename refactoring (but that’s all)
- No semantic highlighting, call hierarchy, etc.
- But protocol is extensible, but each client needs to know about them
CLANGD

- Official part of LLVM project in tree under clang/extras
  - Lots of useful utilities in the core LLVM
- Determines build commands from compile_commands.json files
-Parsed using the clang parser driven by clang compiler driver
  - Works in place of other compilers since clang itself does
- Speeds up parsing by keeping preamble around
  - Precompiles header in memory
- Great diagnostics from clang itself and supports clang fix-its
CLANGD THE BAD

• It has no persistent index! No search
  – The design for one is underway
  – At least they recently added remembering opened files
  – But given that surprisingly useful anyway

• Doesn’t implement LSP “CodeLens” feature

• Releases tied to LLVM/Clang releases

• Hard to extend
  – Adding support for non clang environments
  – new compiler built-ins, include paths, architectures
CQUERY AND CCLS

- cquery is out of tree built against libclang
- Has an index
- Implements all LSP features
- Extends in interesting ways
  - Semantic highlighting, call hierarchy
- But very small community (one person project with a few PRs)
- ccls is a fork of cquery to build in tree
  - Smaller, stronger faster
- Also a one person project
- Neither have considered how to support non-clang environments
CDT LSP4E

- GSOC project this summer by Manish Khurana mentored by Nate Ridge
- Full LSP4E support for both clangd and cquery
- But no support for cquery extensions

- Demo
WHERE DO WE GO FROM HERE?

• Add support for cquery’s extensions
• Work to have those extensions added to the protocol
• Work with clangd to implement those extensions
  – And generally help mature the project
• How do we support non-clang environments
  – Not necessarily upstreamable. Fork?
• Do we work with cquery and/or ccls too?
ADOPT THE NEW IDE ARCHITECTURE

- Language Server takes us down a new path
- Build a Debug Adapter as well for gdb
  - Encourage other debugger integrations to use DAP
- Ensure the LSP4E components for language and debug server are fully featured
  - Parity with existing CDT services
- Continue our effort to support external build systems
  - CMake, Meson, Qt qmake
  - Can we make CDT Managed Build work in the same way
PROVIDE OUR USERS CHOICE

• Help ensure Eclipse is a full partner in the new world
• Take our expertise with C/C++ IDEs and help other IDE platforms
  – Visual Studio Code
  – Theia/Che
  – Others?
• Being a tools developer is about serving the needs of developers
  – CDT has provided the features C/C++ developers need
  – But we need to fit into our user’s workflows
  – Allow them to choose the IDE front end that allows them to be their best
THANK YOU!