Doxygen

Tom Latham
(based on material from Matt Williams)

THE UNIVERSITY OF WARWICK

Documentation

- Documentation is just as important as the code itself
- Without docs, you wouldn't know how to use a library: *cppreference.com* or the *ROOT* docs are essential
- You should make sure you always document your code for external use
- A standard syntax exists called *Doxygen*

Doxygen syntax

- Doxygen comments are generally placed within the header (.hpp) files, rather than the source (.cpp) files

- Doxygen comments are marked in a special way

```
/// Doxygen single-line comments start with three slashes

/**
 * Doxygen multi-line comments start with a slash and two stars
 * In both cases, Doxygen reads what's inside the comment
 */
```

- Comments precede the statement that they want to document

Doxygen commands

- Doxygen provides its own syntax to be used inside comments
- They are detailed in full in the manual but there are a few which are most commonly used:
- The descriptions of functions, classes, enums, etc. often need to be quite detailed, so it's good to also have a short one-liner description that is used at the head of the page – use the `\brief` command
- `\param` is used to document function arguments
- `\return` is used to describe the return value

```
/**
 * \brief This function is amazing
 *
 * More detailed description of all
 * the very special stuff it does
 */
void foo();
```

```
/**
 * \param key the cipher key
 */
void setKey(const size_t key);
```

```
/**
 * \return the number of elements
 */
int size() const;
```

Function example

- Describe the function in a good degree of detail
- Always document all function parameters and return values

```
/**
 * \brief An amazing function which does something very special
 *
 * A longer description of this function is that is can be used
 * to do something very interesting which this longer description
 * explains in detail.
 *
 * \param thing the string that we want to convert
 *
 * \return the converted string
 */
std::string convert(const std::string& thing);
```

Class example

- Class docs should describe the purpose of the class and give examples of usage

```
/**
 * \brief A cipher encodes and decodes
 *
 * Cipher is an abstract base class which provides the ability to
 * encode and decode strings based on a key
 *
 * Use it like
 * \code{.cpp}
   class MyCipher : public Cipher {...};
 * \endcode
 *
 * \since 0.1.3
 */
class Cipher {
```

Enum example

- Enums should have a description and each member should be documented

- A common style is to use 'suffix' comments for the individual member descriptions

```
/**
 * \brief The rank of the employee
 */
enum class Rank {
  Junior, ///< A new person at the company
  Senior, ///< Someone who has been here a while
  Chief   ///< Someone super special
};
```

Separate page example

- Can create pages of documentation that are perhaps not specific to particular classes or functions

- Can create files with .dox extension that use the Doxygen syntax

- Or you can use Markdown files – note that we've simply used our README.md to create the front page of the documentation

```
/**
 * \mainpage Welcome to MPAGS Cipher
 *
 * Blah blah
 *
 * \section Introduction
 * Blah blah
 *
 * \subsection Usage
 *
 * \code{.cpp}
   CaesarCipher c {"4"};
   std::cout << c.applyCipher("test");
 * \endcode
 */
```

Configuring Doxygen

- Doxygen itself is a program that, given a configuration file, generates a set of HTML (or LaTeX, or …) files

- A default configuration file can be created with

```
$ doxygen -G Doxyfile
```

  but we will just use the one that we've provided in the Documentation folder of today's git repository

- It's a simple (if slightly long) file, so feel free to read through it

Automating generation of documentation

Documentation/CMakeLists.txt

```
#Find the Doxygen tools
find_package(Doxygen REQUIRED)

#Copy Doxyfile.in (in source dir) to Doxyfile (in build dir)
#and replace any @VAR@ with with CMake variables called VAR
configure_file( Doxyfile.in Doxyfile @ONLY )

#Tells CMake how to 'create' ${CMAKE_CURRENT_BINARY_DIR}/html/index.html
add_custom_command(
  OUTPUT "${CMAKE_CURRENT_BINARY_DIR}/html/index.html"
  COMMAND ${DOXYGEN_EXECUTABLE}
  WORKING_DIRECTORY ${CMAKE_CURRENT_BINARY_DIR}
  DEPENDS Doxyfile.in ${PROJECT_SOURCE_DIR}/MPAGSCipher ${PROJECT_SOURCE_DIR}/README.md
  COMMENT "Doxygenating ${PROJECT_NAME}"
)

#Adds the ability to do 'make doc' which will try to create ".../html/index.html"
add_custom_target(doc ALL DEPENDS "${CMAKE_CURRENT_BINARY_DIR}/html/index.html")
```

Exercise – build the documentation

- The starter repo for today should contain the necessary files form which to build the documentation
  - Look at what has changed in the top level CMakeLists.txt file
  - Take a look at the new files in the Documentation subdirectory
- Try running "make doc" in your build area
  - Open the resulting file in your web browser: <build_dir>/Documentation/html/index.html
- Can you work out how to have the private members of the CaesarCipher class appear in the documentation?
- Throughout the rest of the day, when adding new code always make sure to document new classes, functions, etc.