Quick Guide to Doxygen

References:

The material in this handout is collected from Doxygen documentation.

Installation

An important part of any project is documentation. Documentation is especially important in group projects. Imagine how useful a browsable index will be when one or more of your team mates are incommunicado and you're trying to figure out their code. This module will require your submissions to be fully documented using Doxygen. Doxygen is an automated documentation formatter and generator for source files implemented in C, C++, and a variety of other programming languages. It allows you to specially tag comments in your source files that will be used to generate nicely formatted HTML and Latex output. This document serves as a brief overview of Doxygen and the features you will use on a regular basis in this module when submitting programming assessments.

Doxygen can generate a variety of graphs including class inheritance diagrams and file dependency graphs. These graphs are generated using an open source graph visualization toolkit called Graphviz. Installing Doxygen and Graphviz and checking if both packages are installed is straightforward in Linux:

```
$ sudo apt-get install -y doxygen graphviz
$ doxygen -v; dot -V
```

Using Doxygen

Step 1: Setting up configuration file

Doxygen requires a configuration file in order to know how you want your documentation to be generated. Each project should get its own configuration file. A project can consist of a single source file, but can also be an entire source tree that is recursively scanned. `doxygen` can create a template configuration file for you. To do this call `doxygen` from the command line with the `-g` option in your project directory:

```
$ doxygen -g <config-file>
```

If you don't provide the file name `config-file`, this command will create a default configuration file `Doxyfile`. This file should be edited a little bit to configure it for the current project's requirements. Open the file in your favorite editor and change the following settings:

- `PROJECT_NAME`: Provide a name like `"Lab 1"` or `"Assignment 1"`, and so on. Don't forget the quotes (because you're providing a string).
- `PROJECT_DESCRIPTION`: Write a one-line description of this project such as `"Test suite for memory debugger"`. Again, don't forget the quotes
- `INPUT`: If you have your source in a directory such as `./src`, you should say so here. Otherwise, leave it empty.
- `OUTPUT_DIRECTORY`: Set this tag to `docs` so Doxygen doesn't clutter your source directory with generated files.

- `TAB_SIZE`: Set this tag to `2` so that code is compact horizontally in the HTML pages.
- `EXTRACT_ALL`: Set this tag to `YES` to make Doxygen assume all entities in the documentation are documented, even if no documentation was available. This is useful if the intent is to collect in one place all the entities (files, classes, functions) in a project even if some of them are undocumented.
- `EXTRACT_PRIVATE`: Set this tag to `YES` to include private members in a `class` or `struct` definition to be included in the documentation.
- `EXTRACT_PRIV_VIRTUAL`: Set this tag to `YES` to include documented private virtual methods of a `class` definition to be included in the documentation.
- `EXTRACT_STATIC`: Set this tag to `YES` to include all static members of a file will be included in the documentation.
- `QUIET`: Set this tag to `YES` so that Doxygen is not chatty and you can view diagnostic messages, if any.
- `SOURCE_BROWSER`: Set this tag to `YES` so that documentation of an entity such as a function can be cross referenced with its definition (source code). This makes analyzing code easier.
- `INLINE_SOURCES`: Set this tag to `YES` so that body of functions, classes, and enums is directly inserted into the documentation.
- `REFERENCED_BY_RELATION`: Set this tag to `YES` so that for each documented entity all documented functions referencing it will be listed.
- `REFERENCES_RELATION`: Set this tag to `YES` so that for each documented function all documented entities called/used by that function will be listed.
- `GENERATE_LATEX`: Set this tag to `NO` because this module is only interested in HTML output.

Step 2: Documenting code

For each entity (file, namespace, class, function) in the code there are two (or in some cases three) types of descriptions, which together form the documentation for that entity; a *brief* description and *detailed* description, both are optional. For both class member and non-member functions there is also a third type of description, the so called *in body* description, which consists of the concatenation of all comment blocks found within the body of the method or function.

There are several ways to mark a comment block as a detailed description. This document will use the Qt style and add an exclamation mark (`!`) after the opening of a C-style comment:

```
/*!
 * ... text ...
 */
```

or

```
/*!
 ... text ...
*/
```

A brief description is a short one-liner and there are several ways to document them. This document will use the `\brief` command with the Qt style comment block shown above. This command ends at the end of a paragraph, so the detailed description follows after an empty line:

```
/*! \brief Brief description.
 *         Brief description continued.
 *
 *  Detailed description starts here.
 */
```

The second option is to use a special C++ style comment which does not span more than one line:

```
/// Brief description.
/** Detailed description. */
```

This is a third option:

```
//! Brief description.

//! Detailed description
//! starts here.
```

All comment blocks can be augmented with HTML tags (to provide links for example).

File headers

To document a file header, the *minimum* tags required are:

- `\file`: Name the source file.
- `\author`: Provide your name so that everybody knows you authored the fie.
- `\date`: Provide date on which file was authored or a special date as the submission date.
- `\par`: Used for free-form information such as course title, section number, assignment title, and so on.
- `\brief`: Provide a one-line description of the file (why? what?).
- More detailed description: An empty line must separate this detailed description and the `\brief` line. It is used to list the entities (classes, enums, functions, class member functions) in the file and anything else of interest.

The following example shows a file header:

```
/*!*****************************************************************************
\file    scantext.c
\author  Nigel Tufnel
\par     DP email: tap11\@digipen.edu
\par     Course: CS120
\par     Programming Assignment #6
\date    11-30-2018

\brief
  This source file implements functions to transform C-style strings.

  The functions include:
  - mystrlen
      Calculates the length (in characters) of a given string.
  - count_tabs
      Takes a string and counts the amount of tabs within.
  - substitute_char
      Takes a string and replaces every instance of a certain character
      with another given character.
```

```
  - calculate_lengths
      Calculates length (in characters) of a given string first with tabs,
      and again after tabs have been converted to a given amount of spaces.

  - count_words
      Takes a string and counts the amount of words inside.
*****************************************************************************/
```

Function header

To document a function header, the *minimum* tags required are:

- `\brief`: Provides a one-line description of the function (why? what?).
- More detailed description: An empty line must separate this detailed description and the `\brief` line. You must describe the purpose of the function. You must publish the algorithm, the pre-conditions (assumptions made by the function in terms of its inputs to execute the algorithm correctly) and post-conditions (what has the function done that might require intervention by the caller such as expecting the caller to free memory that was dynamically allocated by the function).
- \param: Provides information for each parameter. Omitted if there are no parameters. The `\param` command has an optional attribute, `dir`, specifying the direction of the parameter. Possible values are `[in]`, `[in,out]`, and `[out]`.
- `\return`: Document the return value. Omitted if there is no return.
- `\exception`: Document the exceptions that are thrown. C++ only. Exceptions will be covered later in the semester.

The following example shows a function header:

```
/*!*****************************************************************************
\brief
   Replaces each instance of a given character in a string with
   other given characters.

\param[in, out] string
  The string to walk through and replace characters in.

\param[in] old_char
  The original character that will be replaced in the string.

\param[in] new_char
  The character used to replace the old characters

\return
  The amount of characters changed in the string.
*****************************************************************************/
```

Class header

Note that in C++ code, it may be cleaner to describe the class using a \class tag, rather than to put the information in the file header comment. An example of a class header (interface) file `stack.h` and the corresponding source file containing member definitions (implementation file) can be downloaded from the course web page.

Other entities

In addition to `\class` command, other structural commands are:

- `\struct` to document a C-struct.
- `\union` to document a union.
- `\enum` to document an enumeration type.
- `\def` to document a `#define`.
- `\typedef` to document a type definition.
- `\file` to document a file (this is discussed here).
- `\namespace` to document a namespace.

The entire list of special commands can be found here.

Step 3: Running `doxygen`

To generate the documentation, you run the following command

```
$ doxygen <config-file>
```

`doxygen` may generate diagnostic message if it finds inconsistencies which you should fix. Based on the settings in Step 1, `doxygen` will generate browsable HTML documentation in directory `./docs/html`. Navigate to this directory in Windows and browse the documentation by loading `index.html`.