Importance of Documentation of code

- Better understanding of code in bigger teams where peer review of code is done.
- Maintenance of records for quality control.
- In Mission critical and life critical software, documentation is necessary for getting certification like FAA in case of flight software or FDA in case of medical software.
What is Doxygen?

Doxygen is the de facto standard tool for generating documentation from C++ sources, but it also supports other popular programming languages such as C, Objective-C, C#, PHP, Java, Python, IDL (Corba, Microsoft, and UNO/OpenOffice flavors), Fortran, VHDL, Tcl, and to some extent D.
Doxygen can help you in three ways:

1. It can generate an on-line documentation browser (in HTML) and/or an off-line reference manual (in ) from a set of documented source files. There is also support for generating output in RTF (MS-Word), PostScript, hyperlinked PDF, compressed HTML, and Unix man pages. The documentation is extracted directly from the sources, which makes it much easier to keep the documentation consistent with the source code.

2. You can configure doxygen to extract the code structure from undocumented source files. This is very useful to quickly find your way in large source distributions. Doxygen can also visualize the relations between the various elements by means of include dependency graphs, inheritance diagrams, and collaboration diagrams, which are all generated automatically.

3. You can also use doxygen for creating normal documentation (as I did for the doxygen user manual and web-site).
Installation

1. Download the binaries from doxygen website

2. Run configure
   
   `Sudo ./configure`

3. Run make install
   
   `Sudo make`
   
   `Sudo make install`
Creating a Config file

We need to create a config file to create documentation. Doxygen provides a way to auto-generate a config file.

Doxygen \(-g \ <file-name>\)

If you open the file there are several Tags, we have to set Tags of our choice to generate the documentation.
# Doxygen 1.8.10

# This file describes the settings to be used by the documentation system
doxygen (www.doxygen.org) for a project.
#
# All text after a double hash (##) is considered a comment and is placed in
# front of the TAG it is preceding.
#
# All text after a single hash (#) is considered a comment and will be ignored.
# The format is:
# TAG = value [value, ...]
# For lists, items can also be appended using:
# TAG += value [value, ...]
# Values that contain spaces should be placed between quotes (" ").

# Project related configuration options

# This tag specifies the encoding used for all characters in the config file
# that follow. The default is UTF-8 which is also the encoding used for all text
# before the first occurrence of this tag. Doxygen uses libiconv (or the iconv
# built into libc) for the transcoding. See http://www.gnu.org/software/libiconv
# for the list of possible encodings.
# The default value is: UTF-8.

DOXYFILE_ENCODING = UTF-8

# The PROJECT_NAME tag is a single word (or a sequence of words surrounded by
# double-quotes, unless you are using Doxywizard) that should identify the
# project for which the documentation is generated. This name is used in the
# title of most generated pages and in a few other places.
# The default value is: My Project.
Few Important Tags

- OUTPUT_DIRECTORY
- INPUT
- FILE_PATTERNS
- RECURSIVE
- EXTRACT_ALL
- EXTRACT_PRIVATE
- EXTRACT_STATIC
Commenting style for Doxygen

- **Brief Description:** Use a single-line C++ comment, or use the `\brief` tag.

- **Detailed Description:** Use JavaDoc-style commenting `/** ... test ... */` (note the two asterisks `[*]` in the beginning) or the Qt-style `/*! ... text ... */`.

- **In-body Description:** Individual C++ elements like classes, structures, unions, and namespaces have their own tags, such as `\class`, `\struct`, `\union`, and `\namespace`. 
Creating documentation

- To Create a documentation by simply running the following command:
  
  Doxygen <config-file>
Example (configure file)

# Header file to include in order to use Doxygen. If not defined, only the name of
# the header file containing the class definition is used. Otherwise one should
# specify the list of include paths that are normally passed to the compiler
# using the -I flag.

STRIP_FROM_INC_PATH =

# If the SHORT_NAMES tag is set to YES, doxygen will generate much shorter (but
# less readable) file names. This can be useful if your file systems doesn't
# support long names like on DOS, Mac, or CD-ROM.
# The default value is: NO.

SHORT_NAMES = NO

# If the JAVADOC_AUTOBRIEF tag is set to YES then doxygen will interpret the
# first line (until the first dot) of a Javadoc-style comment as the brief
# description. If set to NO, the Javadoc-style will behave just like regular Qt-
# style comments (thus requiring an explicit @brief command for a brief
# description.)
# The default value is: NO.

JAVADOC_AUTOBRIEF = YES

# If the QT_AUTOBRIEF tag is set to YES then doxygen will interpret the first
# line (until the first dot) of a Qt-style comment as the brief description. If
# set to NO, the Qt-style will behave just like regular Qt-style comments (thus
# requiring an explicit \brief command for a brief description.)
# The default value is: NO.

QT_AUTOBRIEF = YES

# The MULTILINE_CPP_IS_BRIEF tag can be set to YES to make doxygen treat a
# multi-line C++ special comment block (i.e. a block of /// or /// comments) as
# a brief description. This used to be the default behavior. The new default is
# to treat a multi-line C++ comment block as a detailed description. Set this
# tag to YES if you prefer the old behavior instead.
#
C Code

/*
 * brief Lab1 Code for switch debouncing
 */

/* This program increments a binary display of the number of button pushes on switch S1 on the mega128 board. */
#include <avr/io.h>
#include <util/delay.h>

uint8_t upper;  /* Variable for digit and ten's place*/
uint8_t lower;  /* Variable for digit and unit's place*/

/*******************************************************************************************************/
// debounce_switch
// Adapted from Ganssel's "Guide to Debouncing"
// Checks the state of pushbutton S1. It shifts in ones till the button is pushed.
// Function returns a 1 only once per debounced button push so a debounce and toggle function can be implemented at the same time. Expects active low pushbutton on
// Port D bit zero. Debounce time is determined by external loop delay times 12.
/*******************************************************************************************************/
uint8_t debounce_switch() {
    static uint16_t state = 0; // holds present state
    state = (state << 1) | (!bit_is_clear(PIND, 0)) | 0x0000;
    if (state == 0xF000) return 1;
    return 0;
}

/*******************************************************************************************************/
// Get digits at ten's and unit place and convert that into BCD
/*******************************************************************************************************/
uint8_t get_bcd(uint8_t tens, uint8_t unit)
{
    tens = tens<<4;
    return tens|unit;
}

/*******************************************************************************************************/
lab1_code.c File Reference

```c
#include <avr/io.h>
#include <util/delay.h>
```

### Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int8_t debounce_switch()</td>
<td>Variable for digit and unit's place. More...</td>
</tr>
<tr>
<td>uint8_t get_bcd(uint8_t tens, uint8_t unit)</td>
<td>Get digits at ten's and unit place and convert that into BCD More...</td>
</tr>
<tr>
<td>int main()</td>
<td>Check switch S1. More...</td>
</tr>
</tbody>
</table>

### Variables

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8_t</td>
<td>lower</td>
<td>Variable for digit and ten's place. More...</td>
</tr>
<tr>
<td>uint8_t</td>
<td>upper</td>
<td>Used for switch debouncing. More...</td>
</tr>
</tbody>
</table>
Variable for digit and unit \textquotesingle{}s place.
Reference:


Project.tar.gz