CS 6015: Software Engineering

Spring 2024

Lecture 9: Documentation

Last Week

- Defensive programming
- Test/Code Coverage

This Week

- Documentation (paired with Lab 4 – due tomorrow)
- Let Binding (Project related)
- Parsing (Project related)
- Code review today
- Assignment 5 released – due next Tuesday

Code style: Example 1

```
/* Use the insertion sort technique to sort the "data" array in ascending order.
This routine assumes that data[ firstElement ] is not the first element in data and
that data[ firstElement-1 ] can be accessed. */ public void InsertionSort( int[]
data, int firstElement, int lastElement ) { /* Replace element at lower boundary
with an element guaranteed to be first in a sorted list. */ int lowerBoundary =
data[ firstElement-1 ]; data[ firstElement-1 ] = SORT_MIN; /* The elements in
positions firstElement through sortBoundary-1 are always sorted. In each pass
through the loop, sortBoundary is increased, and the element at the position of the
new sortBoundary probably isn't in its sorted place in the array, so it's inserted
into the proper place somewhere between firstElement and sortBoundary. */ for (
int sortBoundary = firstElement+1; sortBoundary <= lastElement; sortBoundary++ )
{ int insertVal = data[ sortBoundary ]; int insertPos = sortBoundary; while (
insertVal < data[ insertPos-1 ] ) { data[ insertPos ] = data[ insertPos-1 ];
insertPos = insertPos-1; } data[ insertPos ] = insertVal; } /* Replace original
lower-boundary element */ data[ firstElement-1 ] = lowerBoundary; }
```

Code style: Example 2

```
/* Use the insertion sort technique to sort the "data" array in ascending
first element in data and that data[ firstElement-1 ] can be accessed. */
public void InsertionSort( int[] data, int firstElement, int lastElement ) {
/* Replace element at lower boundary with an element guaranteed to be first in a*/
int lowerBoundary = data[ firstElement-1 ];
data[ firstElement-1 ] = SORT_MIN;
/* The elements in positions firstElement through sortBoundary-1 are between firstElement and sortBoundary. */
for (
int sortBoundary = firstElement+1;
sortBoundary <= lastElement;
sortBoundary++
) {
int insertVal = data[ sortBoundary ];
int insertPos = sortBoundary;
while ( insertVal < data[ insertPos-1 ] ) {
data[ insertPos ] = data[ insertPos-1 ];
insertPos = insertPos-1;
}
data[ insertPos ] = insertVal;
}
/* Replace original lower-boundary element */
data[ firstElement-1 ] = lowerBoundary;
}
```

Code style: Example 3

```
/* Use the insertion sort technique to sort the "data" array in ascending
order. This routine assumes that data[ firstElement ] is not the
first element in data and that data[ firstElement-1 ] can be accessed.*/

public void InsertionSort( int[] data, int firstElement, int lastElement ) {
     // Replace element at lower boundary
     int lowerBoundary = data[ firstElement-1 ];
    data[ firstElement-1 ] = SORT_MIN;

    /* The elements in positions firstElement through sortBoundary */
    for ( int sortBoundary = firstElement + 1; sortBoundary <= lastElement; sortBoundary++ ) {
        int insertVal = data[ sortBoundary ];
        int insertPos = sortBoundary;
        while ( insertVal < data[ insertPos - 1 ] ) {
            data[ insertPos ] = data[ insertPos - 1 ];
            insertPos = insertPos - 1;
            }
        data[ insertPos ] = insertVal;
        }

    // Replace original lower-boundary element
    data[ firstElement - 1 ] = lowerBoundary;
}
```

Objectives of good layout

- Represent the logical structure of the code
- Improve readability
- Ease maintenance

Layouts: More

```
for ( int i = 0; i < MAX_LINES; i++ )
{
 ReadLine( i );
 ProcessLine( i );
}
```

vs

```
for ( int i = 0; i < MAX_LINES; i++ )
   {
      ReadLine( i );
      ProcessLine( i );
   }
```

Layouts: More

```
if ((('0' <= inChar) && (inChar <= '9')) || (('a' <= inChar) &&
   (inChar <= 'z')) || (('A' <= inChar) && (inChar <= 'Z')))
    ...
```

vs

```
if ( ( ( '0' <= inChar ) && ( inChar <= '9' ) ) ||
   ( ( 'a' <= inChar ) && ( inChar <= 'z' ) ) ||
   ( ( 'A' <= inChar ) && ( inChar <= 'Z' ) ) )
   ...
```

Layouts: More

```
customerBill = PreviousBalance( paymentHistory[ customerID ] ) + LateCharge
   ( paymentHistory[ customerID ] );
```

vs

```
customerBill = PreviousBalance( paymentHistory[ customerID ] ) +
   LateCharge( paymentHistory[ customerID ] );
```

Layouts: More

```
totalBill = totalBill
   + customerPurchases[ customerID ]
   + CitySalesTax( customerPurchases[ customerID ] )
   + StateSalesTax( customerPurchases[ customerID ] )
   + FootballStadiumTax()
   - SalesTaxExemption( customerPurchases[ customerID ] );
```

Keys to effective comments

```
// write out the sums 1..n for all n from 1 to num
current = 1;
previous = 0;
sum = 1;
for ( int i = 0; i < num; i++ ) {
    cout << "Sum = " <<  sum << endl;
    sum = current + previous;
    previous = current;
    current = sum;
}
```

Keys to effective comments

```
// set product to "base"
product = base;


// loop from 2 to "num"
for ( int i = 2; i <= num; i++ ) {
    // multiply "base" by "product"
    product = product * base;

}
cout << "Product = " << product << endl;
```

Keys to effective comments

- Use styles that don’t break down or discourage modification
- Use the design recipe steps to reduce commenting time
- Integrate commenting into your development style
- Performance is not a good reason to avoid commenting

Documentation

- External documentation
  - Unit-development folders (UDF) or Software-development folder (SDF)
  - Detailed documentation
- Internal documentation

Documentation Tools

- In-program documentation contained in the comments can also be used to generate other forms of documentation

- Why documentation tools?
  - Rewriting the document could lead to inconsistency between external and in-program documents
  - Rewriting documents takes more time and resources

Comments & Guidelines

- Source files with the *main* function:
  - The purpose of the program
  - The version number
- Other source files or classes (header files):
  - The author
  - Revision history
  - A description of the class

Comments & Guidelines

- Functions
  - Explanation of the function usage
  - Explanation of how the function works
  - Description of parameters
  - Description of return value and special return values
  - What exceptions can be thrown
- Member Functions of Classes
  - Dependencies if any

Comments & Guidelines

- Statements Blocks
  - Explanation of nonobvious functionality
  - Provide insight into subtleties in the algorithm
  - Explain why the statements do what they do

Doxygen

- *Annotates programs written in C++, C, C#, PHP, Java, …*
- Extracts documentation directly from the sources
  - Easier to preserve documentation consistency with source code
- Extracts the code structure from undocumented source files
  - Generates and visualizes the relations between elements: dependency graphs, inheritance diagrams, …
- Generates documentation in the form of html, pdf, as well as other forms
- Can be used on Linux, Windows, & Mac
- Many projects use doxygen. Ref: https://www.doxygen.nl/projects.html

How to use Doxygen

Step1 : Create a configuration file

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

- One per project
- Determines all settings of doxygen
- Consists of a number of assignments (tags) with default values <TAGNAME> = <VALUE>
- Has a default name *Doxyfile*

Configuration File Snapshot

Configuration file template

Description of the tag field

```
#---------------------------------------------------------------------------
# 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 tex
# 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/libicon
# 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.

PROJECT_NAME           = "My Project"

# The PROJECT_NUMBER tag can be used to enter a project or revision number. Thi
# could be handy for archiving the generated documentation or if some version
# control system is used.

PROJECT_NUMBER         =

# Using the PROJECT_BRIEF tag one can provide an optional one line description
# for a project that appears at the top of each page and should give viewer a
# quick idea about the purpose of the project. Keep the description short.

PROJECT_BRIEF          =

# With the PROJECT_LOGO tag one can specify an logo or icon that is included in
# the documentation. The maximum height of the logo should not exceed 55 pixels
# and the maximum width should not exceed 200 pixels. Doxygen will copy the log
# to the output directory.
```

Configuration File Tags

- **<OUTPUT_DIRECTORY>:** You must provide a directory name
- **<EXTRACT_ALL>:** This tag is an indicator to doxygen to extract documentation even when the individual classes or functions are undocumented. You must set this tag to **Yes**.
- **<EXTRACT_PRIVATE>:** Set this tag to **Yes**. Otherwise, private data members of a class would not be included in the documentation.
- **<EXCLUDE>:** Add the names of the files and directories for which documentation should not be generated separated by spaces.

How to use Doxygen

Step 2: Document your code

Step 3: Run Doxygen

```
doxygen  <config-file-name>
```

- Generates documentation files according to set settings in configuration file
- Default output director is the directory in which Doxygen is started
- By default, doxygen generates documentation in Latex and HTML formats.
- Other formats can be added by setting the value of their tags to YES

Comment Blocks

Provides detailed description

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

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

Brief command

- A special command that allows a brief description
- 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.
*/
```

Main Page Documentation

```
/**
* \mainpage MSDScript
* \author First Author
* \author Second Author (if applicable)
* \date 06-02-2024
*/
```

- Customizes the index page
- Place it before a class or *above the main()*

Function Documentation

```
/**
* \brief Calculates distance between two points
* \param x1 first argument, x position of first point
* \param x2 second argument, x position of second point
* \return double value of distance
*/
Alternative for \param:
double distance(double x1 //!< first point x value
, double x2 //!< second point x value
) ;
```

- Can be omitted for void functions and constructors

Class Documentation

- Class inheritance hierarchy diagram is automatically generated by Doxygen
- Brief description before the class
- Comments for member methods (same as before)

Class Members Documentation

- Brief description

```
int var; //!< Brief description after the member
int var; ///< Brief description after the member
```

- Detailed description

```
int var; /*!< Detailed description after the member */
int var; /**< Detailed description after the member */
```

- The < marker indicates that the member is in front of the block instead of after the block

File Documentation

```
/**
* \file expression.cpp
* \brief contains expression class
definition
*
*
* More verbose description here
* \author Author Name
*/
```

```
/**
* \file expression.h
* \brief expression class
*
* Longer description goes here.
*/
```

At the beginning of files

Other Commands

- Documentation in other formats such as pdf
- Reference: https://www.doxygen.nl/index.html