A Standard for Java/C++ Code (with minor modifications)

Prof. Timothy A. Gonsalves

Distributed Systems & Optical Networks Lab

TeNeT Group

Department of Computer Science & Engineering

I.I.T., Madras - 600 036

17 February, 1999

Preamble

========

The use of uniform coding and documentation standards makes it easier

for one person to read and understand the code written by another.

This is important in a group project, or when a program is likely to

be maintained and/or enhanced by someone other than the original

author(s).

Names

=====

To make the code self-documenting, choose meaningful names for

variables. Abbreviations may be used so long as they are widely

accepted and do not hinder intelligibility. A good test of names is:

can you read your code to a fellow programmer over the phone?

Package names should be capitalized the same way as the

product/organization name. E.g.:

package TeNeT.CygNet.Alarm;

For names that consist of multiple words, capitalize the first letter

of each word, do not use '\_'. Distinguish classes of names as fol-

lows:

Classes: First letter uppercase

Methods, Functions: First letter lowercase

Variables: First letter lowercase

Constants: All uppercase, separate words with '\_'

Some examples:

class AnExampleClass

public void showExample()

int x, y, roomMessDistance[MAX\_ROOMS]

AnExampleClass firstExample

public static final int MAX\_LINE\_LEN 500

Remember, a name is typically typed a few times, and read many times.

If you do not like typing, use the power of a text-editor to ease your

burden: type an abbreviation, then do a global search-and-replace; or

use the automatic abbreviation expansion capabilities of editors such

as Emacs.

Names should differ in more than one character, especially if they are

of the same type. E.g., txBuf and rxBuf differ in only the first

character which occurs on adjacent keys on the keyboard. txBuf and

rcvBuf would be a better choice.

For class and object names, the use of the suffix Cls and Obj is

optional. Use it if it enhances readability.

Use the following to identify particular names:

Type a defined type (e.g. class MsgType {...})

Cls A class name

Obj An object of a class

Ptr Pointer (e.g. bufPtr, msgPtr, pktPtr)

Addr Address (e.g. ioBaseAddr)

Flg Boolean (e.g. moreFlg)

Str String (e.g. promptStr)

Chr Character (e.g. inChr, outChr)

Tx Transmit

Rcv Receive

Buf Buffer (e.g., txBuf, rcvBuf)

Msg Message

Pkt Packet

Tab Table (e.g. relayTab, relayTabPtr)

Num Number (e.g. numPkts) ["No" could be confused with the

negative]

Ctrl Control

Cmd Command

Que Queue (e.g. inBufQuePtr)

Len Length (e.g. pktLen)

Hdr Header (e.g. hdrLen)

Trl Trailer

Grp Group

Internal Documentation

======================

Apart from external documentation such as pseudo-code, flow-charts,

state transition diagrams, function-call hierarchies, and prose, the

program files should contain documentation. Begin each file with a

comment including the following fields:

/\*\*

\* MyClass.java: A 1-line description of the class

\*

\* Purpose:

\* Several lines of text giving the purpose in some detail

\*

\* @author Your Name

\* @version $Id$

\* @see CygNet.Alarm.AlarmParser

\* @see java.awt.Button

\*

\* Chronological list of all major changes and

1

\* bug fixes

\* Revision history:

\* A. Programmer, 7/7/77

\* released version 1.0

\*

\* C. Debugger, 8/8/88

\* fixed stack overflow with null input

\*

\* Eager B. Eaver, 9/9/99

\* added aNewProc() to support 3-D

\*

\* Bugs, tests to be done, etc

\* Comments:

\* The program occasionally crashes when two users

\* access the database simultaneously during the

\* new moon.

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Declare each variable (except temporaries such as loop indices) on a

separate line, followed by an inline comment explaining the purpose of

the variable. Use

char rcvState; // of the receive FSM

char txState; // of the transmit FSM

rather than

char rcvState, // of the receive FSM

txState; // of the transmit FSM

Where appropriate, group variables in blocks by function, and

alphabetically within each group.

Preceding each function, include a comment block as follows:

/\*\*

\* getString - get a string from the moon.

\*

\* @param bufSize is the size of the buffer.

\* @return the string

\* @exception IOException if the moon is not visible

\*

\* Method: a brief description if necessary.

\*

\* Bugs: if any

\*

\* To be done: if anything

\*

\*/

String getString(int bufSize)

{

...

} /\* End of getString() \*/

Within the body of the function, on separate lines at the start of

each major block, describe briefly the purpose and peculiarities of

the block. For obscure statements, include an inline comment. Avoid

obvious comments such as:

i++; /\* increment i \*/

Layout

======

Indent the code according to the following scheme. Each new block

should be indented 4 spaces from the previous one. Open/close braces

should be in the same column and both indented 2 spaces from the

previous level. Label the closing brace of a large block with a

comment. Use blank lines liberally to indicate breaks in the flow of

control. Use lines longer than 80 columns only infrequently.

Continuation lines for multi-line statements should be indented. In a

sequence of assignment statements, line up the '=' signs.

/\* The main loop, terminates when done \*/

while (moreFlg)

{

if (i == 2)

doSomethingAppropriate();

else

doSomethingElse();

System.out.println("This is the result\n");

/\* Mumbo-jumbo for each file \*/

for (j = 0; j < maxFile; j++)

{

total += table[i].wordCount;

i = j + 1;

count = j - 1;

}

} /\* while (moreFlg) \*/

External Documentation

======================

As you work, prepare and keep up-to-date the following for each

package/class/method that you work on (some may not be relevant for

simple functions that do not call other functions). All documents and

code must contain your name and date of each significant revision.

1. Specification - inputs, purpose, side-effects, outputs

2. Implementation - details on major data structures, pseudo-code

for algorithms, highlighting anything unusual, to-be-done and

known bugs or limitations

3. Test plan - either exhaustive or selective with justification

4. Test log - each time the function/module is "released" for use

by others, the log must contain an entry certifying that you

have run the complete test plan. During testing, do not use

other untested modules.

5. List of files with a one-line description of each

6. List of constants, types, variables and functions with a one-

line description of each and the file in which it is defined.

Indicate whether each is PUBLIC, PRIVATE (to a file) or LOCAL

(to a function).

7. Function call-tree, i.e., for each function, a list of all

functions that it calls (use the Microsoft calltree utility or

equivalent).

8. Function called-by tree, i.e., for each function, a list of all

functions that call it (use the Microsoft calltree utility or

equivalent).

Revision Control

================

Each software package is to have a version number consisting of four

parts, e.g., 2.3.15c. The first three are mandatory, the fourth is

optional.

Major version This should change only when significant new features

are added to the software. The change must be agreed

upon by all members of the development group. E.g.,

2.3.15c has major version 2.

Minor version This should change when minor new features are added

to the software. The change must be agreed upon by

all members of the development group. E.g., 2.3.15c

has minor version 3. Note that 2.3.10 is a later

version than 2.3.9.

Patch level This is incremented every time a bug is fixed. This

can be incremented by the individual applying the

patch. E.g., 2.3.15c has patch level 15.

Environment code

The software may run under different environments

such as OS, computer, etc. Alternatively, there may

be different versions for different customers. Each

such environment is identified by a unique letter.

E.g., 2.3.15c has environment code c. Note:

environment code is optional.

We recommend the use of the Concurrent Version System (CVS) package for

maintaining different versions, especially with multi-member teams.

CVS is available at no cost and runs on Unix, Windows and other

systems.

Homily

======

Make it a practice to include documentation on the above lines as you

write the code for the first time. Adding documentation later is much

more tedious and prone to error (you may no longer be quite sure why

you did something months ago) and often gets short shrift in the rush

to get the last bug out before the deadline for completing the

project.

------------------------- Footnote

1. For multi-file programs, it may be better to have the revision

history entirely in one file (such as the one containing main()).