Coding Standard

Author: Yan Li

Email: [yan\_li@brown.edu](mailto:yan_li@brown.edu)

Introduction

This document talks about the coding standard that I use. It’s illustrated via C++ example but most standards applies to Java/C as well.

File header

All files shall have the file header like the following format, no matter they are source files or header files. It may not be totally same as the following depends on the programming language, but it must have general information about the name of the file, description, author and date or time stamp:

|  |
| --- |
| /\*  @file:  @desc:  @author:  @date:  \*/ |

Structure/Class header

Structure or Class must also have a header describes the functionality of it. It could be like the following format:

/\*

@class:

@brief:

\*/

|  |
| --- |
|  |

Variables Naming Conventions

In C: the local variables should be separated by “\_”, like: my\_variable; the global variables should begin with “g” like: g\_global

In C++/Java: the local variables should begin with lower-case letters and all of the following words should begin with upper-case like: myLocalVariable

Functions Naming Conventions

The name of functions has the same rule as local variables in each programming language.

Macros Naming Conventions

All letters in Macros should always be upper-case. Each word must be separated by “\_”, like: BUFFER\_SIZE

Members Naming Conventions

Member variable should begin with “\_” or “m\_”

Enumerations Naming Convention

The enumeration name must always be upper case letters; each word must be separated by “\_”, like:

enum TYPE

{

POLYGON,

POINT,

LINE

}

Class/Structure/Interface Naming Conventions

Each word in the name of class/structure/interface should always begin with upper-case letter.

Class Definitions

All classes shall declare all public, protected, and private members, in that order. For each section, member functions shall precede member variables.

|  |
| --- |
| class MyClass  {  public:      MyClass();      ~MyClass();    protected:      void    protectedFunc();      int   m\_myProtected;    private:      void    myPrivateFunc();      int   m\_myPrivate;  }; |

Constructor (C++/Java)

The class constructor shall initialize all members of the class. In Java, it may be initialized in

Class’s member list. In that case no initialization need to be done in the constructor. In

The constructor shall not perform any dynamic allocations in the constructor in C++. All classes that require dynamic allocations shall do so using another function.

Provide a copy constructor and assignment operator only when necessary in C++.

Destructor (C++)

The class destructor shall release all pointers to objects that the class has acquired, or any object the class has created.

|  |
| --- |
|  |

File Structure (C/C++)

All class files shall be given the same file name for a given class. For example: "MyClass.h", "MyClass.cpp, “MyClass.c”

Header Files (C/C++)

The class definition shall be contained the class header file.

The class header file shall include only those header files required to compile the header file. Function and class prototypes shall be used instead of file includes whenever possible.

Source Files

The class implementation shall be contained in the class source file.

The order of the inline functions shall match the order in which they are declared in the class definition (C/C++).

Function Header

All functions except for those short or non-important functions shall have the following header:

|  |
| --- |
| /\*\*  \* @brief  \* @param  \* @param  \* @param  \* @return  \*/ |

Statement Length

Statements should not exceed the 80-character line-limit. It's probably alright to exceed 80 columns occasionally.

White Space

Use spaces to make logical expression readable.

|  |
| --- |
| while((numVertices < maxVertices ) &&         ((numIndices + 3 ) < maxIndices)) |

Use spaces to make arithmetic readable.

int result = a + b \* c;

|  |
| --- |
|  |

Use spaces to make routine arguments readable.

|  |
| --- |
| sendData(type, number, buffer); |

Use spaces to align groups of related assignments.

|  |
| --- |
| numberThreads  = 5;  maximumThreads = 10;    numberProcesses  = 6;  maximumProcesses = 12; |

Continuation

Make the incompleteness of a statement obvious.

|  |
| --- |
| while((numVertices < maxVertices ) &&         ((numIndices + 3 ) < maxIndices))  {      total =          numVertices +          numIndices;  } |

Keep closely related elements together.

|  |
| --- |
| size =      sizeof(SlowState) +      sizeof(FastState); |

Use alignment in function signature

|  |
| --- |
| sendData(type,           number,           buffer); |

Logical structure should be separated if using multiple lines.

|  |
| --- |
| if(((a > b) &&      (b < c)) ||      (d > e)) |

Indent assignment continuation lines after the assignment operator.

|  |
| --- |
| totalSize =      sizeof( SlowState ) +      sizeof( FastState ); |

Code-Blocks

If-statements in C/C++:

|  |
| --- |
| if (condition)  {      statements;  }  if (condition)  {      statements;  }  else if (condition)  {      statements;  }  else  {      statements;  }  In Java the left parenthesis is after “)”, this applies to while, for as well:  if (condition){      statements;  } |

While-loops in C/C++:

|  |
| --- |
| while(condition)  {      statements;    } |

|  |
| --- |
| do  {      statements;    }while (condition); |

For-loops in C/C++:

|  |
| --- |
| for (init value; condition; statement)  {      statements;  } |

Switch-statements:

|  |
| --- |
| switch (value)  {  case 0:      statements;      break;  case 1:      statements;      break;  case 2:      statements;      break;  default:      statements;      break;  }; |