CAMPUSTALK

Coding Convention

**Team-5**

**IT-632 Software Engineering**

**Instructor- Prof. Asim Banerjee**

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Author(s) | Date | Description |
| 1.0 | Sunny Lalwani |  | Created |
|  |  |  |  |

**Content**

**1. Introduction**

**2. Internal Documentation Standard**

**3. Coding Standards**

3.1 Indentation

3.2 Class and function declaration

3.3 Statements

3.3.1 “if", "if else", "if else-if else" Statements

3.3.2 "for" Statements

3.3.3 "while" Statements

**4. Blank Spaces**

**5. Error Handling**

**6. Comments and variable names**

1. **Introduction**

Coding standards are the rules which all the programmers need to follow instead of writing the code in his own preferred style. Coding Standards will ensure that different coders will follow the same pattern and same style of coding. It becomes very easy for every group member to follow it as well as to integrate the whole project created by different people. It will maintain consistency as well as integrity of the whole code through-out the project. The use of these guidelines should result in readable code and should encourage adherence to them. It also ensures that any developer who looks at the code will know what to expect throughout the entire application

Well written software offers many advantages. It will contain fewer bugs and will run more efficiently than poorly written programs. Since software has a life cycle and much of which revolves around maintenance, it will be easier for the original developer(s) and future keepers of the code to maintain and modify the software as needed. This will lead to increased productivity of the developer(s). The overall cost of the software is greatly reduced when the code is developed and maintained according to software standards.

The standards (Standards are rules which programmers are expected to follow) we have chosen are all down to personal preference and what we found easiest to code and read.

**2. Internal documentation standards**

If done correctly, internal documentation improves the readability of a software module.

We are focused on using good internal documentation practices.

Each module contained within the source file should be preceded by a block of comments showing the following:

* The name of the module.
* The name of the original author.
* The date the module was created.
* A description of what the module does.
* A list of the calling arguments, their types, and brief explanations of what they do.
* A list of required files and/or database tables needed by the routine, indicating if the routine expects the database or files to be already opened
* Return values

**3. Coding Standards**

**3.1 Indentation**

Proper and consistent indentation is important in producing easy to read and maintainable programs. A tab (four columns or four spaces) is the basic unit of indentation.

For eg:

Function samplefunction()

{

/<tab>/

code;

code;

if (indentAnotherLevel)

{

/<tab>/

moreCode;

moreCode;

if (indentAnotherLevel)

{

/<tab>/

moreCode;

moreCode;

}

}

}

**3.2 Class and Function Declarations**

When coding classes and functions, the following formatting rules are followed:

* No space between a method name and the parenthesis "(" starting its parameter list.
* Open brace "{" appears at the end of the same line as the declaration statement.
* Closing brace "}" starts a line by itself indented to match its corresponding. opening statement, except when it is a null statement the "}" should appear immediately after the "{".

**3.3 Statements**

Each line contains at most one statement. Example:

string scriptname; /\* Followed \*/ int userid; /\* Followed\*/

int userid; string scriptname; /\* Not Followed \*/

**3.3.1 "if", "if else", "if else-if else" Statements**

The if-else class of statements are defined in the following form:

**Syntax 1-**

if (condition)

single statement

else

single statement

**Syntax 2-**

if (condition)

{

statements;

}

**Syntax 3-**

if (condition)

{

statements;

}

else

{

statements;

}

**Syntax 4-**

if (condition)

{

statements;

}

else if (condition)

{

statements;

}

else

{

statements;

}

**3.3.2 "for" Statements**

A for statement is given the following form:

for (initialization; condition; update)

{

statements;

}

**3.3.3 "while" Statements**

A while statement is given the following form:

**Syntax 1-**

Initialization;

while ( condition)

{

Statements;

Update;

}

**Syntax 2-**

while (initialization; condition; update)

{

statements;

}

**4. Blank Spaces**

Blank spaces are used in the following circumstances:

A keyword followed by a parenthesis should be separated by a space. Note that a blank space is not used between a method name and its opening parenthesis. This helps to distinguish keywords from function calls. A blank space appears after commas in argument lists. All binary operators except '.' is separated from their operands by spaces. Blank spaces never separate unary operators such as unary minus, increment ("++"), and decrement ("--") from their operands. Example:

a += c + d;

a = (a + b) / (c \* d);

The expressions in a for statement is separated by blank spaces.

Example:

for (expr1; expr2; expr3)

Casts are followed by a blank space.

Examples:

myFunction((int) a, (int) b);

**5. Error Handling**

Use of try, catch blocks should be done wherever there is possibility of run time error generation like floating point error or failure in connecting to a database. In catch blocks proper simple error message must be displayed without giving technical details about the error to the user.

Example-

try {

body-code

} catch (exception-classname variable-name) {

handler-code

}

**6. Comments**

appropriate by the developer in the following way:

// Populate language menu in a consistent way

Code;

/\*Comments\*/

**Variable Names**

1. Wherever possible, keep variable names to fewer than 15 characters, although you can sacrifice a few extra characters to improve clarity.

2. Names should be descriptive, but also concise.

3. There's no hard and fast rule when it comes to the length of a variable name, so just try and be as concise as possible without affecting clarity too much.

4. Avoid all caps and all lowercase names. Single lowercase words or letters are acceptable.

5. Do not create namespaces, classes, methods, properties, fields, or parameters that vary only by capitalization.

6. Do not use names that begin with a numeric character.

7. Variables and Properties should describe an entity not the type or size.

8. Avoid using abbreviations unless the full name is excessive.

9. Avoid abbreviations longer than 5 characters.

10. Any Abbreviations must be widely known and accepted.

11. Use uppercase for two-letter abbreviations, and Pascal Case for longer abbreviations.

12. Avoid adding redundant or meaningless prefixes and suffixes to identifiers