E-Gram Seva

Coding Standard Document v1.0

Team 22
April 1, 2013

REVISION HISTORY

Version	Author	Date
Version 1	Rutvik Jhala	April 1 , 2013
Version 1 Review	Karan Makim	April 8, 2013

CONTENTS

1. Introduction

The goal of these guidelines is to create uniform coding habits among software personnel in the engineering department so that reading, checking, and maintaining code written by different persons becomes easier. The intent of these standards is to define a natural style and consistency, yet leave to the authors of the engineering department source code, the freedom to practice their craft without unnecessary burden. Not only does this solution make the code easier to understand, it also ensures that any developer who looks at the code will know what to expect throughout the entire application.

When a project adheres to common standards many good things can happen like:

- Programmers can go into any code and figure out what's going on, so maintainability, readability, and reusability are increased. Walking through code becomes less painful.
- New people can get up to speed quickly.
- People new to a language are spared the need to develop a personal style and defend it.
- People make fewer mistakes in consistent environments.

2. Standards

2.1 Indentation

A consistent use of indentation makes code more readable and errors easier to detect. Indentation should be used to:

- Emphasize the body of a control statement such as a loop or a select statement.
- Emphasize the body of a conditional statement.
- Emphasize a new scope block.

A tab (four columns or four spaces) is the basic unit of indentation. Once the programmer chooses the number of spaces to indent by, then it is important that this indentation amount be consistently applied throughout the program.

For Example

```
Function myFunction() {
  /<tab>/

    code;
   code;
if (indentAnotherLevel) {
    /<tab>/
  more Code;
  more Code;
  if (indentAnotherLevel) {
```

```
/<tab>/
 more Code;
 more Code;
}
}
}
```

2.2 Classes, Subroutines, Functions and Methods:

Keep subroutines, functions, and methods reasonably sized. This depends upon the language been used. When coding classes and functions, the following formatting rules are followed:

- No space between a method name and parenthesis “(” starting its parameter list.
- Open brace “{” appear 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 “{”.

The names of the classes, subroutines, functions and method shall have verbs in them. That is names shall specify an action.
For example: “get_name”, “compute_size”

2.3 Statements:

Each line contains at most one statement.
Example:

```
intdirlen; /* Correct */
stringasmbase; /* Correct */
intdirlen;
stringasmbase; /* AVOIDED! */
```

2.3.1 "if", "if else", "if else-if else" Statements

```
if(condition)  single statement;  else single statement;

if(condition)

{
        statements;
}
if(condition)
{
        statements;
}
else {
statements;
 }
```

```
if(condition)
{
        statements;
}
else if(condition)
{
        statements;
}
else {
        statements;  }
```

2.3.2 “for” Statements:

A for statement is given the following form:

```
for (initialization; condition; update)
 {
        statements;
 }
```

2.4 Queries Statements

Queries written in our document is in the form given below:

- SELECT * from diseasedb where id='3';
- INSERT INTO test.diseasedb (id,name,symptoms,cure)

It is case-sensitive.

2.5 Use of Braces

Braces are used to delimit the bodies of conditional statements, control constructs and the blank of scope. Programmer should use the following bracing style:

```
for (int j = 0 ; j< max_iterations;++j)
 {
    /* Some work is done here*/
 }
```

The above bracing style is more readable and leads to neater looking code. Make sure that style is consistent throughout the code. When code is written by different author, adopt the style of bracing mentioned above.
Braces shall be used even when there is only one statement in control block. For example:
The above bracing style is more readable and leads to neater-looking code. Make sure that style is consistent throughout the code. When editing code written by different author, adopt the style of bracing used.
Bad:

```
  if (j == 0)
printf (“j is zero.\n”);
```

Better:

```
if (j == 0)
{
printf (“j is zero.\n”);
}
```

2.6 Comments

No specific format shall be followed for comments. In general trailing comments or End-of-Line comments shall be used. The comments have been used wherever deemed appropriate by the developer in the following way:

// populate language menu in a consistent way
code;

3. Guidelines

3.1 Spacing

The proper use of spaces within a line of code can enhance readability. Good rules of thumb are as follows:

- A keyword followed by a parenthesis should be separated by a space.
- A blank space should appear after each comma in an argument list.
- All binary operator s except “.” Should be separated from their operands by spaces. Blank spaces should never separate unary operators such as unary minus, increment (“++”), and decrement (“--”) from their operands.

Example:

Bad

```
cost=price+(price*sales_tax); fprintf(stdout ,“The total cost
is %5.2f\n”,cost);
```

Better

```
cost = price + ( price * sales_tax );
 fprintf(stdout,“The total cost is %5.2f\n”, cost) ;
```

3.2 Wrapping Lines

When an expression will not fit on a single line, break it according to these following principles:

- Break after comma:

Example

```
fprintf ( stdout , “\nThere are %d reasons to use standards\n” ,
num_reasons ) ;
```

- Break after an operator:

Example

```
longinttotal_apples = num_my_apples + num_his_apples +
num_her_apples ;
```

3.3 Variable Declarations

Variable declarations that span multiple lines should always be preceded by a type.

Example:
Acceptable:

```
int price , score ;
```

Acceptable:

```
int price ;  int score ;
```

Not Acceptable:

```
int price ,
score ;
```

4. References

They are as follows:

www.nws.noaa.gov/oh/hrl/...docs/General_Software_Standards.pdf
www.cse.buffalo.edu/~rapaport/code.documentation.Excerpts.pdf