Coding Standards and Good Programming Practices

Table of Contents

1. Naming Conventions

1. Camel Casing: First characters of all words, except the first word are Upper Case and other characters are lower case. Use Camel casing for methods and variables names. For example,

```
void FeetAndInchesToMetersAndCent (…)
{
//Keep all first letters capital for Functions
Int sumOfQuizzes;     //keep first letter small for first word in variables
Int sumOfAssignments;
…
}
```

2. Do not use Hungarian notation to name variables. For example,

```
int nAge;
```

3. Identifiers for constant should be in uppercase letters. For example,

```
const double CONVERSION = 2.54;
const int NO_OF_STUDENTS = 20;
const char BLANK = ' ';
```

4. Use Meaningful, descriptive words to name variables. Do not use abbreviations.

   Good:

```
string address
int salary
```

   Not Good:

```
string addr
int sal
```

5. Do not use single character variable names like i, n, s etc. Use names like index, temp. One exception in this case would be variables used for iterations in loops:

```
for ( int i = 0; i < count; i++ )
{
        ...
```

```
}
```

2. Indentation and Spacing

1. Use TAB for indentation. Do not use SPACES. Define the Tab size as 4.
2. Curly braces ({}) should be in the same level as the code outside the braces.

```
if (beta >= 10)
{
        int alpha = 10;
        beta = beta + alpha;
        cout << alpha << ' ' << beta << endl;
}
```

3. Use one blank line to separate logical groups of code.

Good:

```
int main()
{
        double one, two;

        cout << "Enter two numbers: ";
        cin >> one >> two;
        cout << endl;

        traceMe(one, two);
        traceMe(two, one);

        return 0;
}
```

Not Good:

```
int main()
{
        double one, two;
        cout << "Enter two numbers: ";
        cin >> one >> two;
        cout << endl;
        traceMe(one, two);
        traceMe(two, one);
        return 0;
}
```

4. The curly braces should be on a separate line and not in the same line as if, for etc.
   Good:

```
for (counter = 0; counter < 5; counter++)
{
        cin >> item[counter];
        sum = sum + item[counter];
}
```

   Not Good:

```
for (counter = 0; counter < 5; counter++){
        cin >> item[counter];
        sum = sum + item[counter];
}
```

5. Use a single space before and after each operator and brackets.

   Good:

```
if (hours > 40.0)
{
        …
}
```

   Not Good:

```
if (hours>40.0)
{
        …
}
```

3. Good Programming practices

1. Avoid writing very long methods. A method should typically have 1~25 lines of code. If a method has more than 25 lines of code, you must consider re factoring into separate methods.
2. Method name should tell what it does. Do not use misleading names. If the method name is obvious, there is no need of documentation explaining what the method does.

   Good:

```
void SavePhoneNumber ( string phoneNumber )
```

```
{
        // Save the phone number.
}
```

Not Good:

```
// This method will save the phone number.
void SaveDetails ( string phoneNumber )
{
        // Save the phone number.
}
```

3. A method should do only 'one job'.
4. Always watch for unexpected values.
5. Do not hardcode numbers. Use constants instead. Declare constant in the top of the file and use it in your code.

4. Comments

Good and meaningful comments make code more maintainable. However,

1. Do not write comments for every line of code and every variable declared.
2. Use // or /// for comments. Avoid using /* … */
3. Write comments wherever required. But good readable code will require very less comments. If all variables and method names are meaningful, that would make the code very readable and will not need many comments.
4. Do not write comments if the code is easily understandable without comment. The drawback of having lot of comments is, if you change the code and forget to change the comment, it will lead to more confusion.
5. Fewer lines of comments will make the code more elegant. But if the code is not clean/readable and there are less comments, that is worse.
6. If you have to use some complex or weird logic for any reason, document it very well with sufficient comments.
7. The bottom line is, write clean, readable code such a way that it doesn't need any comments to understand.