A C# Programming Style Guideline

Most of a programmer's efforts are aimed at the development of correct and efficient programs. But the readability of programs is also important. There are many "standards" in use. Since following them will not guarantee good code, these "standards" are actually guidelines of style. Each one has its proponents and detractors. The textbook advocates one particular style for C# programs. This document collects the style comments in the textbook into one place and adds some documentation requirements for this course. The only significant difference between this document and the textbook is that for the purposes of making programs easier to grade, it asks that all variable declarations be given at the beginning of a function. The essential point is that a program is a medium of communication between humans; a clear, consistent style will make it that much easier for you to communicate.

Program Documentation

1. Program Heading

The main program in a console application and each new class declaration should begin with a comment section in the following form and with the following information filled in:

Example

```
  namespace YourNameSpace
    {
     // File: <name of file>
     // Your name                                          Date completed
     // Engr 123
     // Assignment: <name and number of assignment>
     // < Description of what the program does >
     //
     // -------------------------------------------------------------------
        public class YourClass : System.Windows.Forms.Form
          {
```

2. Function Headings

Include documentation, as needed, of the following form for each function:

Example:

```
//  private double MyFunction(double myVariable)
//  This routine ...<tell what the function does>
//  List the input parameters and the output variables.
//  List any assumptions about the state of the input variables such as
//    variable myVariable must be an double between 0 and 100.
    private double MyFunction(double myVariable)
      {int i, numPoints;
```

3. Identifiers

Identifiers should be chosen to be self-documenting. Abbreviations, except where standard, should be avoided. In addition, comment variables when usage is restricted.

Each word in a function identifier, class identifier, or structure type identifier should start with an uppercase letter. e.g., FindMinIndex or Point.

Each word except the first one in an variable identifier should start with an uppercase letter. e.g., firstName.

Constant identifiers should be written in all uppercase with words separated by an underscore or dollar sign if needed for clarity. e.g., MAX_STRING_SIZE.

4. Comments
Comment code to improve clarity. Comments should tell WHAT is being done or why it is being done, not how it is being done. For example,

```
  // Adjust i to point to the end of the previous word:
     i = i - 1
is better than

  // Subtract 1 from i:
     i = i - 1
```

Often these comments will be phrases found in the program design.
Comments should be in good English. Grammar and spelling should be correct.

Variables and Constants
1. Constants should be declared globally if there is a possibility of using them in more than one function, otherwise they should be declared locally to the function that uses it. However, while there are valid reasons for having global variables, they generally should be used sparingly.

2. Each (non-class attribute) variable identifier that occurs in a function should be local to that function - that is, declared in the function's header or declaration part.

(a) If the variable may have its value changed in the body of the function and that new value will be needed back in the calling program (i.e., it is passed back), then the variable should be declared as a reference formal parameter.

(b) If the variable gets its initial value from the calling program but does not send a different value back (i.e., it is only received), then the variable should be declared as a value formal parameter, except for C# arrays, which are automatically passed as reference parameters.

(c) If the variable does not get its initial value from the calling program and does not pass its value back (via a parameter), then the variable should be declared as a local variable of the function. This generally includes the returned object, if any.

3. Constants in your algorithm should generally be replaced by constant identifiers in your program. Exceptions should be made when the constant conveys its own meaning, such as 0 as an initial value for a sum or 1 to start a count, or is part of a constant mathematical formula, such as 2 in x*x + 2*x + 3.

Program Formatting
1. All local constants and local variables should be declared at the beginning of a function or main program before any executable statements. (Note, this is different than most C# guidelines, which recommend that variables be declared just before first use. The reason for this guideline in this course is that it makes it easier for the instructor to grade programs.)

2. A blank line should be used to separate declarations from the executable statements. In general, blank lines should be used wherever their use will improve readability.

3. Indenting should be used to convey structure. Indentation should be at least 2 spaces, and generally no more that 5 spaces unless otherwise noted. Indenting should be consistent. Do not use tabs for indenting since programs must be turned in and tab settings on the grader's machine are not likely to be the same as on your machine. Use the options menu in visual studio to set the text editor tabs to spaces.

4. A statement should not be longer than a screen's width (about 80 characters). If a non-I/O, non-function call statement must be continued on more than one line, the second line should be indented to at least 6 spaces and successive continuation lines should be aligned with the second line of the statement. For example, we might write

```
while ((('a' <= line [i]) && (line[i] <= 'z'))
      || (('A' <= line[i]) && (line[i] <= 'Z')))
   i++;
```

A function call statement should be broken up so that the function arguments lined up. For example, we might write

```
Sample (argument1, argument2, argument3,
        argument4, argument5, argument6);
```

5. For the statement that follows if, else, while, for, do, and switch: The statement should start on the next line and be indented. For example,

```
if (first <= last)
     found = true;
```

6. Column alignment should be observed for each pair of curly braces. When a compound statement is needed, the curly braces should line up under the first line for that statement. Each line of the body of a compound statement should be indented. For example,

```
if (a [middle] = item)
  {item     = a [middle];
   found    = true;
   position = middle;
  }  // end if match found
```

7. Column alignment should be observed for each set of reserved words if and else. This include multi-branch constructs, for example:

```
if (x > 90)
   grade = 'A';
else if (x > 80)
   grade = 'B';
else if (x > 70)
   grade = 'C';
else if (x > 60)
   grade = 'C';
else
   grade = 'F';
```

8. Comments that describe one or more statements should be immediately above and aligned with the statement or collection of statements which they describe. There should be a blank line before such comments. For example,

```
j = i;
while ((j > 1) && (a [j - 1] > a [j]))
  {
   // a [1..j-1] is unsorted and a [j..i] sorted:
   Swap (a [j], a [j - 1]);
   j = j - 1
  } // end while
```

9. At least one space should be used in the following locations within C# text (this does not apply within comments and character strings):

(a) before and after =, //, any relational, logical, arithmetic, or assignment operator
(b) before (
(c) after a comma in argument lists, and after semicolon in for loop headings

10. A function should fit on one screen (about 25 lines) if possible and must fit on a listing page (about 50 lines). Ideally, the analysis and design will not produce code that is more than a page long, but if the code does not fit initially, introduce one or more new functions to split up the work in logical places.