VSFX 375 – Coding Standards

(adapted from ITGM 315 – Dean Lawson
modified for VSFX 495 now 375 – Deborah R. Fowler)

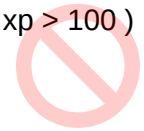
All code written for this class whether in class or for an assignment must conform to this document. This is a living document and may be updated as needed throughout the course.

1. **Variable and Function Names**
    a. All multiple word names must have each successive word start with a capital letter – example: “localVariable”
    b. Names should avoid abbreviations – example: use “transformMatrix” rather than “tm”
    c. Variable names should begin with a lower case character – example: “localVariable”
    d. Function names should begin with a lower case letter – example: “myFunction”
    e. Class names should start with an upper case letter – example: “MyClass”
    f. Class member variables should begin with a lower case ‘m’ – example: “mHitPoints”
    g. Constants and Enums should be in all upper case with underscores separating words – example: “PI” or “MAX_SCORE”
    h. Global functions and variables should be avoided. However, if used they should be prefaced with a lower case ‘g’ – example: “gCurrentScore”
    i. Pointers should be prefaced with a lower case ‘p’ and references should be prefaced with a lower case ‘r’ – examples: “pUnit”, “ppUnit”, “rUnit”

2. **Magic Numbers**

    a. **Do not use numbers directly in your code**. Instead use a const variable. For instance:

```
if ( xp > 100 )
{
}
```

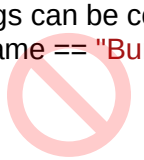


```
//is much less understandable than:

const int XP_LEVEL2 = 100;
if ( xp > XP_LEVEL_2 )
{
}
```

    b. Strings can be considered Magic Numbers as well:

```
if ( name == "Burt" )
{
}
```



```
//is much less understandable than:
const string MONSTER_NAME = "Burt";
if ( name == MONSTER_NAME )
{
}
```

3. **Code formatting**

   a. **Matching braces should line up** – example:

```
if  (  localVariable == 3 )
{
        // code inside should be indented
}
else
{
        // code inside should be indented
        // this makes it easy to visually match braces
}
```

   DO NOT USE:

```
if  (  localVariable == 3 ){
// code
}
else {
}
```

   b. Tabs should be 4 spaces.

   c. Use **White space** to make code more readable - example:

```
while ( maxHitPoints > 0 )
{
        if ( x == 1 )
        {
                RotateObject();
                InvertObject();

                maxHitspoints++;
        }
        else
        {
                maxHitPoints--;
        }

        RedrawScreen();
}
```

DO NOT USE:

```
while(maxHitPoints>0)
{
        if(x==1)
        {
                RotateObject();
                InvertObject();
                maxHitspoints++;
        }
        else
        {
                maxHitPoints--;
        }
        RedrawScreen();
}
```

Also when using a do-while loop please be sure to put space after the curly brace:

```
do
{
        // a bunch of code here
} while ( some condition );
```

d. It is legal in C++ to not use curly braces at all if there is only a single command following an if, else, while or for. **Use them anyway**. That way, if you later want to add a command, you do not have to remember to add curly braces too.

4. Commenting

Proper use of // or /* symbols for comments are fine.

a. **Each function (including main()) should contain a comment block** explaining the purpose of the function, describing the input parameters and any output. Using Example:

```
/*
        GetNumTemples - function to return the number of
    temples owned by a particular god

        Parameters:
                inGodNum - index of the god

        Return:
                int - the number of temples owned by the passed
                in god
*/
int GetNumTemples( int32 inGodNum )…
```

b. If a function uses a particularly confusing algorithm then it should be described in the function header comment block

```
/*
    CalcDerivedData - function to calculate the data derived
  from the passed in Unit

    Parameters:
    unit - a reference to the unit to derive data for

    Return:
    Data - a new Data class containing all derived data

    This function uses the following algorithm to calc data:
    1) Get the birthdate of the unit
    2) Multiply birthdate by the current number of hit points
    3) Divide by PI to adjust for rounding errors
*/
    Data CalcDerivedData( Unit& unit )...
```

c. Comment lines in the function where something might be less than obvious to the code reader.

```
        // temp will be used to assist in switching the
        // values of x and y
        int temp = x;
        x = y;
        y = temp;
```

d. Do not use comments to restate the code.
DO NOT do this:

```
int currentHitPoints = 0; //set currentHitPoints to 0
currentHitPoints++; //increment the currentHitPoints
currentHitPoints /= 2;//divide the currentHitPoints by 2
```

e. For code blocks that are long enough that they do not fit on a single screen and force the reader to scroll, add an end-of-line comment following the final curly brace, denoting what code block it is closing:

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

        // assume there is a lot of code here

 } // end for i
```