**Review and Research**

You can review the video of Session 2 here:

<https://blizzard.sharepoint.com/portals/hub/_layouts/15/PointPublishing.aspx?app=video&p=p&chid=8aa7fa80-bfa2-4021-bf44-543dba93f693&vid=9cac298d-adc0-4377-ba3b-ae89fe09ef84>

As always, email me at [semerson@blizzard.com](mailto:semerson@blizzard.com) if you have any questions!

A review of if-statements:

<http://csharp.net-tutorials.com/basics/if-statement/>

Here is a good review of loops. This tutorial also explores the “foreach” loop, which we’ll be discussing in class when we begin to talk about data structures:

<http://csharp.net-tutorials.com/basics/loops/>

If-then statements can sometimes become rather complex; in certain cases a “switch” statement may be better. Here’s a tutorial to show you how to use it:

<http://csharp.net-tutorials.com/basics/switch-statement/>

**Glossary**

If-then statement – A statement in a program that allows execution to branch along different paths on the basis of specific conditions being met.

Loop – A structure within a program that allows a block of code to repeat until specific conditions are met. Sometimes a loop may bypass execution altogether, and it’s possible in most programming languages (including C#) to write a loop that never terminates; be careful not to do this!

Prefix operator / Postfix operator – When an operator is applied *before* a variable, we call it a prefix operator (e.g., ++x). If the operator is applied *after* the variable, we call it a postfix operator (e.g., x++). A lot of common mathematical operators are of the infix type, which means they should be used *between* different variables or values. These include common operations like addition, subtraction, etc.

Const – A variable that is declared ‘const’ in C# cannot be altered once it has been initialized. If you wish to specify unchanging values in your program, such as the maximum number of items on a list or the width of the display, it’s a good idea to declare these values as const. It’s worth noting that various kinds of literals, such as “apple” or the number ‘3’ are considered const by C#.

Magic Numbers – In CS we often refer to numeric literals that appear in a program without a clear context as Magic Numbers. You should almost always prefer to replace a Magic Number with a clearly-named variable, possibly declared const. For example, if I see this:

float cost = numberOfItems \* 3.75f;

We would consider the ‘3.75’ to be a Magic Number. It’s not clear what it means. Instead, try this:

const float pricePerUnit = 3.75f;

// Somewhere later in the code…

float cost = numberOfItems \* pricePerUnit;

This is much clearer. The cost is obtained by multiplying the number of items by the price. If you see Magic Numbers in your code consider replacing them with named values instead.

Type conversion / Typecasting – Explicitly converting a value of one type to another type; i.e., converting an int to a float or a float to a double.

Narrowing conversion – A typecast that may cause a value to lose data. For example, casting a float to an integer is considered a narrowing conversion, because an integer cannot represent the decimal component of the float. Implicit narrowing conversions (that is, narrowing conversions that you don’t explicitly ask for) are usually disallowed by C#.

**Practice Exercises**

These exercises are grouped around concepts we’ve covered in class and range from very simple to more complex. As always, a few reminders:

1. When compiling and running these exercises in Visual Studio, be sure to do so in Debug Mode. You can do this with the hotkey combination ctl+F5.
2. These exercises are meant to be a start to your practice, but if you want more ideas, contact me and let me know. I’ll be happy to make additional suggestions!
3. Don’t be afraid to repeat an exercise several times. Repetition in your practice is very helpful, especially in the beginning when you’re trying to get comfortable with syntax as well as new concepts.
4. If you get stuck on a problem or don’t understand why something is happening, please contact me and let me know. I’ll be happy to help.
5. An exercise may occasionally require you to use a concept we haven’t covered in class; when this is the case, the exercise will be marked with an asterisk there will be a link to research you can do help you find the new information you need.

*Functions and control flow*

Ex. 1. Write a function that takes three floats as input. Return true if all three values are positive; return false if any of them are zero or negative.

Ex. 2. Write a function that takes three characters as input. The function should write the three characters to the console in alphabetical order. HINT: It may be helpful to convert all characters to lower-case to deal with mixed input. You can do this with the char.ToLower() function. Here’s an example of how to use it:

char c = ‘A’ // c is initialized to an upper-case ‘A’

c = char.ToLower(c) // This will change c to a lower-case ‘a’

Ex. 3. Write a function that prints the numbers 1 through 10, inclusive, to the console. Modify the function so that it does this five times in a row.

Ex. 4. Write a function that, given the floating-point width and length of a rectangle, returns the area. If the function is called with invalid input (e.g., if either value is less-than or equal-to zero) then print a warning message to the console and return 0f.

Ex. 5. Write a function that, given the floating-point radius of a circle, returns the approximate area of the circle. Use the value 3.14 to approximate the constant pi. Avoid magic numbers!

Ex 6. Complete the following function:

static void PrintWithDecoration(string message, char ornament, int count)

{

// Some code goes here…

}

The function should print the string ‘message’ to the console with a decoration on either side. The decoration should be comprised of ‘ornament’ repeated ‘count’ times. So the following call:

PrintWithDecoration(“Hello”, ‘\*’, 5)

Should print the following to the console:

\*\*\*\*\*Hello\*\*\*\*\*

Ex. 7. Write a function that, given a range of positive integers, returns the first one that is both odd and divisible by seven. You can use the remainder operator ‘%’ to help you with this. If there is no valid return value in the supplied range, return 0.

Ex. 8. Imagine you own a shop that sells widgets. Each widget costs $2.00 to manufacture, and will be sold for $3.50. Write functions with the following names to perform the given tasks (in each case use floating-point numbers to represent dollar amounts, e.g., $3.50 can be represented as 3.5f):

1. TotalCost: Determine the total cost of manufacturing n widgets.
2. GrossProfit: Determine the gross profit of n widgets sold (i.e., the total amount of money you’ll make in sales minus the total cost of manufacturing the widgets.)
3. RequiredWidgets: Given a floating point number that represents a gross target profit, determine the minimum number of widgets that will need to be sold to achieve the target profit.

Check your functions against the following input / output:

TotalCost(20) // Should return 40f

GrossProfit(30) // Should return 45f

RequiredWidgets(50.00f) // Should return 34

…Now change the cost of each widget to $1.75 and the sale price to $4.15. The new outputs should be:

TotalCost(20) // Should return 35f

GrossProfit(30) // Should return 72f

RequiredWidgets(50.00f) // Should return 21

(If you avoided magic numbers in your implementation, that second set of tests should be very easy!)

Ex 9. Write a function that, given a positive integer ‘max’ as input, writes the Fibonacci Sequence to the console up to ‘max.’ If you’re unfamiliar with the Fibonacci Sequence, here’s a site where you can get a simple definition as well as some other interesting information:

<https://www.mathsisfun.com/numbers/fibonacci-sequence.html>