**CS109 Sample Questions for the Final**

These questions are pulled from several previous finals. Your actual final will include fewer questions and somewhat less programming and you would not be asked to write programs that use objects, but should be able to read programs that utilize objects. However, these problems should give you an idea of what type of questions you will be asked.

**1. Short Answer.** Provide brief (1-3 sentence) answers to the following:

a) What is the role of a stub in modular program design?

b) How many array entries are created by the following statement?

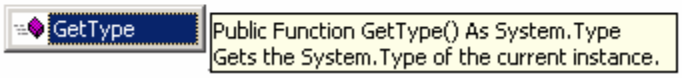
```
Dim a(10) As Integer
```

c) Given the following class:

```
Public Class Foo
    Public x As Integer
End Class
```

When we enter the following code and type the "." VB.NET pops up:

```
Dim a As New ArrayList
Dim f As New Foo
a.Add(f)     ' Add to arraylist
a(0).
```




Why does only GetType appear and not "x" when "x" is a public member of the object?

d) In the previous problem in part c, why does "GetType" appear when it does not look like we defined such a property in class Foo?

e) Describe what a breakpoint is and how the debugging tool in Visual Studio can be used to inspect the contents of variables.

f) What is the difference between the radio button control and the checkbox control?

## 2. Find the Bugs

All of the following code snippets have one or more bugs. Identify each one and fix the bug. Assume the necessary code is in place to make a working program (e.g. the form exists, the code is within some event, etc.).

a) The following should continue until the user enters a number from 1-3:

```
Dim i As Integer = -1
While ((i <> 1) Or (i <> 2) Or (i <> 3))
    i = CInt(InputBox("Enter either 1, 2, or 3"))
End While
```

b) The following code should print “Big” or “Bigger”

```
Dim i As Integer
i = CInt(InputBox("Enter a integer."))
If (0 < i < 10) Then
    Console.WriteLine("Big")
ElseIf (10 < i) Then
    Console.WriteLine("Bigger")
End If
```

c) The code below should swap the values of X and Y, outputting “2 1”

```
Dim x, y As Integer
x = 1
y = 2
Swap(x, y)
Console.WriteLine(x & " " & y)
```

The swap subroutine is defined as follows:

```
Public Sub Swap(ByRef x As Integer, ByRef y As Integer)
    x = y
    y = x
End Sub
```

d) In the course notes for Selection Sort, I erroneously gave you the following buggy code:

```
Sub SelectionSort(ByVal ary() As Integer)
    Dim j, i As Integer
    Dim temp As Integer
    Dim minValue As Integer
    Dim minPos As Integer
    Dim lastIndex As Integer

    lastIndex = ary.Length - 1

    For j = 0 To lastIndex - 1
        ' Find the index of the smallest element
        ' between position j and lastIndex.
        ' Start by assuming the min is in position j+1
        minValue = ary(j + 1)
        minPos = j + 1
        For i = j + 2 To lastIndex
            If ary(i) < minValue Then
                ' New minimum
                minPos = i
                minValue = ary(i)
            End If
        Next
        ' Swap min with position j
        temp = ary(j)
        ary(j) = ary(minPos)
        ary(minPos) = temp
    Next
End Sub
```

Give an example of an input array where the code will not correctly sort the array as it is written.

### 3. Nested Loops

Rewrite the following code such that the functionality remains the same, but uses two while loops instead of the for loops.

```
    Dim i, j, n As Integer

    n = 11
    For i = 11 To 1 Step -2
        For j = 1 To i
            Console.Write("*")
        Next
        Console.WriteLine()
    Next
```

**4. Loops and Subroutines: Implementing the "choose" function**

a) The factorial of a nonnegative integer *n* is written *n!* and is defined as follows:

n! = n* (n-1) * (n-2) * … 3 * 2 * 1
and
n! = 1 when n = 0 or n = 1

e.g.:
1! is equal to 1,
2! is equal to 2*1 = 2
3! is equal to 3*2*1 = 6
4! is equal to 4*3*2*1 = 24, etc.

Write a function called **factorial** that returns the factorial, as an integer, of the number passed in (also as an integer).

b) Written $\binom{m}{n}$ and pronounced "*m choose n*", the *choose* function is how many different ways *m* objects can be chosen from a collection of *n* objects. For example, for the set of numbers {1,2,3} n=3.

If m=1 then we have three ways of choosing one item from the set: {1}, {2}, {3}.
If m=2 then we have three ways of choosing two items from the set: {1,2} {1,3}, {2,3}.
If m=3 then we have one way of choosing three items from the set: {1,2,3}.

The choose function has many applications in statistics and in the analysis of algorithms. *m choose n* is defined as:

$$\binom{m}{n} = \frac{n!}{(m!)((n-m)!)}$$

Write a function called **choose** that returns m choose n. The function should take as input two integer parameters for m and n and return the answer as an integer. The choose function should invoke the factorial function you wrote in part a.

c) Write code that could go into a button-click or other event that displays the number of the ways we could select items out of a set that varies from 1 to 5 objects. Here is a sample output illustrating the desired behavior:

1 objects can be chosen from 1 objects 1 ways.

1 objects can be chosen from 2 objects 2 ways.
2 objects can be chosen from 2 objects 1 ways.

1 objects can be chosen from 3 objects 3 ways.
2 objects can be chosen from 3 objects 3 ways.
3 objects can be chosen from 3 objects 1 ways.

1 objects can be chosen from 4 objects 4 ways.
2 objects can be chosen from 4 objects 6 ways.
3 objects can be chosen from 4 objects 4 ways.
4 objects can be chosen from 4 objects 1 ways.

1 objects can be chosen from 5 objects 5 ways.
2 objects can be chosen from 5 objects 10 ways.
3 objects can be chosen from 5 objects 10 ways.
4 objects can be chosen from 5 objects 5 ways.
5 objects can be chosen from 5 objects 1 ways.

## 5. Arrays

Given the following array declaration:

```
Dim intArray() As Integer = {55, 12, 103, 120, 98, 5, 109, 991, 3, 59}
```

Write code that scans through the array and outputs the largest and the smallest value contained in the array.

## 6. Classes and Inheritance : Calculating Postage

The amount that it costs to ship a package varies depending on its size and weight.

a) Write a class called **Package**. The class should have:
- A member variable to store the weight of the package in pounds as a double
- A property called "Weight" that allows a user to get and set the weight variable
- An overridable function called "CalcShipping" that determines the cost to ship the package using the formula of *cost* = $3 per pound. The calculated cost value should be returned by the function.

b) Create a derived class called **Box** for box packages. Box should inherit from Package. The class should have:
   - Member variables to store the dimensions of the box's height, width, and depth in inches stored as doubles.
   - Properties named "Height", "Width" and "Depth" to get and set the member variables
   - Override the CalcShipping function, where if the height + width + depth is greater than 108 inches, then $30 is added to the normal cost for a package (where normal cost is what CalcShipping returns as computed in the Package class). If the height + width + depth is not greater than 108 inches then the function should return the normal cost for the package and not add $30.

c) Write sample code that could go into a button-click event that creates a box, assigns a weight, height, width, and depth, and prints the shipping cost.

## 7. Classes and Constructors

Given the following class:

```
Public Class MyClass
    Public val As Integer = 1

    Public Sub New()
        val = 2
    End Sub

    Public Sub New(ByVal newNum As Integer)
        val = newNum
    End Sub

    Public Sub Print()
        Console.WriteLine(val)
    End Sub
End Class
```

What would this code output to the Console?

```
Dim r1 As New MyClass
Dim r2 As New MyClass(3)

r1.Print()
r2.Print()
```

## 8. Food Sales

Now that VB.NET is over and summer is here, it means selling food from your booth at the State Fair. You have an array of foods that you sell, e.g.:

```
Dim aryFoods() As String = {"Elephant Ears", _
    "Turkey Leg", "Salmon Taco", "Deep Fried Twinkie"}
```

Each food is given an ID, which is its index in the array. For example, 0 refers to Elephant Ears, 1 refers to Turkey Leg, 2 refers to Salmon Taco, and 3 refers to Deep Fried Twinkies.

At the end of the day you get a log of food sales with the ID of each food in the order it is sold. The log is stored in an array. For example, if you sold, in order: two elephant ears, two turkey legs, one elephant ear, one twinkie, one salmon taco, and one twinkie, then the array would look like this:

```
Dim arySaleLog() As Integer = {0, 0, 1, 1, 0, 3, 2, 3}
```

Write a subroutine called `CountSales` that inputs both arrays as parameters and outputs to the Console window the number of sales for each food.

Given the arrays defined above,

```
CountSales(aryFoods, arySaleLog)
```

Would output:

```
You sold 3 Elephant Ears
You sold 2 Turkey Leg
You sold 1 Salmon Taco
You sold 2 Deep Fried Twinkie
```

Make sure that your subroutine works for any array of foods, not just the example given here. For example, it should also work if there were 10 foods in the array, with ID's from 0 to 9. Hint: Create a separate array to count the number of each food item. Use `aryFoods.Length` to get the number of items in the array.

**9. The Time Machine**

Your time machine is capable of going forward in time up to 24 hours. The machine is configured to jump ahead in minutes. To enter the proper number of minutes into your machine, you would like a program that can take a start time (in hours, minutes, am/pm) and a future time (in hours, minutes, am/pm) and calculate the difference in minutes between the start and future time.

A time is specified in your program with three variables:

```
Dim hours, minutes As Integer
Dim ampm As String
```

For example, to represent 11:50 AM, you would store:

```
hours = 11
minutes = 50
ampm = "AM"
```

This means that to store a start and a future time you need six variables:

```
Dim hrsStart, minStart As Integer
Dim  ampmStart As String
Dim hrsFuture, minFuture As Integer
Dim  ampmFuture As String
```

a) Describe an algorithm that takes a start and future time using the variables described above and computes the difference in minutes. Be specific and give the necessary equations and logic to perform the conversion. Describe your algorithm using pseudocode.

   For example, given a start time of 11:59 AM and a future time of 12:01 PM, your algorithm should compute 2 minutes as the time difference.

b) Write the Visual Basic .NET code to implement your algorithm from part a. Write your code in a function called **ComputeDifference** that takes the six variables as parameters to represent the start time and future time. Your function should return, as an Integer, the time difference in minutes.