Comp 170 Exam 1 Overview.

Resources During the Exam: The exam will be closed book, no calculators. You may bring notes on two sides of 8.5x11 inch paper (either both sides of one sheet, or two sheets written on single sides). I want to test you on concepts, not memorized rote facts. This will also be the pattern for future exams.

Main topics that may be on exam 1

1. Syntax of code in a class, using System, with a static void Main …. Static functions, formal parameters, return values, actual parameters.
2. The ways data come into a function (so far): through parameters or keyboard input.
3. The ways a method leaves an effect after its termination (so far): return value, screen output.
4. Storing intermediate results in local variables.
5. Effect of a return statement; need for return statements; consistency with function heading.
6. Variables and assignment statements. Follow a sequence of steps using the latest values of variables.
7. Boolean conditions: && (and), || (or), ! (not).
8. Simple and compound if statements (reading deeply nested ones, writing simpler ones).
9. Primitive types int, Boolean, and char.
10. Understand int arithmetic operators including / (division without remainder), % (remainder). Be able to evaluate all arithmetic expressions involving ints, with proper precedence.
11. Understand (and recall or be able to look in your notes for!) the methods and property of the string Lab, including the basic idea of a string index; distinction between single and double quotes.
12. Format strings, including specifying the number of digit after the decimal point.
13. String concatenation with +
14. Read while loops; write simple interactive loops and loops following a simple arithmetic sequence.
15. Scope of variables.

How the topics get used:

1. Be able to follow code, one instruction at a time, within a function (static method), if statements, while statements, calling functions (in execution order in sequence and jumping around and repeating, keeping track of current variable values).
2. Be able to write a few lines of code using the above. You will be expected to follow explicit given code that is much more complicated than the code which you need to write.

Conventions:

1. I will always follow standard indenting conventions, so blocks will always be clearly indented, corresponding lines with if's and else's will always line up. I will never try to confuse you with the layout.
2. I try not to test people on speed. I try to leave lots of time for the slowest student. Use an extra period!

Read the following before looking at either the problems or the solutions!

1. Study first, compiling notes, and then look at the sample problems. Look at the list at the top of the page and start by filling in any holes. The sample problems cannot give complete coverage, and if you look at them first, you are likely to study just these points first, and will not get an idea how well you are prepared in general.
2. Do not look at the answers until you have fully studied and tried the problems and gotten help getting over rough spots in the problems if you need it! Looking at the answers before this time makes the problems be just a few more displayed examples, rather than an opportunity to actively learn by doing and check out where you are. The doing is likely to help you be able to do again on a test.

The sample problems start on the next page.

Sample Problems (with answers at the end)
In all requested output, show the output in the order of execution, with line breaks shown. All together, these problems are somewhat longer than an exam.

Always assume we have code with the line `using System;`

1. What would this code fragment print?

```
int a, b;
a = 2;                                  //1
b = 5 +a*(3 + 13 % 3);                  //2
Console.WriteLine(
         "a: " + a + " b: " + b);       //3
a = a + 2*b;                            //4
b = a + b;                              //5
Console.WriteLine(
         "a: " + a + " b: " + b);       //6
```

2. What is printed from each call to the function `Choose`?
   a. Choose(3, 4); b. Choose(5, 4);
   c. Choose(30, 40); d. Choose(50, 40);

```
static void Choose(int x, int y)
{
   if (x < y) {              //1
      Console.Write("A");    //2
   }
   else if (x < 2*y) {       //3
      Console.Write("B");    //4
   }
   if (x + y < 50) {         //5
      Console.Write("C");    //6
   }
   Console.WriteLine("D"); //7
}
```

3. What would this code fragment print?

```
string s = "abcdefghijk";
string t = "Compute";
int n = t.Length;                       //1
string u = s.Substring(2,n);            //2
char ch = t[3];                         //3
Console.WriteLine(u.ToUpper() + ch); //4
```

4. What would this code fragment print?

```
string s = "multiply";      //1
int i = 0;                  //2
while (i < 5) {             //3
   Console.Write (s[i]);    //4
   i = i + 2;               //5
}
```

5. What would this code fragment print?

```
string s = "Here";
double x =  12.468;
int a = 5, b = 10;
Console.WriteLine("{0} are {1:F1} and {2}!", s, x, a+b);
```

6. What is printed from each call to the function TestIt? Follow the nesting of if-else statements.
   a. TestIt(2, 5); b. TestIt(4, 5); c. TestIt(13, 6);
   d. TestIt(12, 22); e. TestIt(5, 4); f. TestIt(1, 2);

```
static void TestIt(int x, int y)
{
   Console.Write(x+" "+y+" ");      //1
   if (x % 2 == 0 || y % 3 == 0) { //2
      if (x < 10 && 3 < x) {        //3
         Console.Write("D");        //4
      }
      else {                        //5
         if (y < 20)  {             //6
            Console.Write("E");     //7
         }
      }
   }
   else {                          //8
      if (x + y > 5) {             //9
         Console.Write("F");       //10
      }
      else {                       //11
         Console.Write("G");       //12
      }
   }
}
```

7. What is displayed by the program Test?

```
class Test
{
   static void Main()
   {
      int x = 1, y = 4;          //1
      int z = foo(x, y);         //2
      x = Foo(y, z);             //3
      Console.WriteLine(
        x + ":" + y + ":" + z);  //4
   }

   static int Foo(int a, int b)  //5
   {
      int c= a + b;              //6
      return 2*c;                //7
   }
}
```

8. Complete the definition of the function `IsLegalTriangle`, with three int parameters, which are the supposed sides of a triangle. The function should return true if the sides are legal and false if they are not legal. In a legal triangle each side is shorter than the sum of the other two sides. For example:
   Sides 3, 4, and 5 are legal: $3 < 4+5$, $4 < 3 + 5$, $5 < 3 + 4$ so `IsLegalTriangle(3, 4, 5)` is true.
   Sides 2, 10, and 6 are not legal: $10 \geq 2+6$ so `IsLegalTriangle(2, 10, 6)` is false.

```
static Boolean IsLegalTriangle(int a, int b, int c)
```

9. Complete the definition of the function `FirstHalf`, which returns the first half of its string parameter. If the string has an odd number of characters, the middle character should not be included. (This is the simplest way to deal with an odd number.) For example:
   `FirstHalf("12345678")` returns `"1234"`
   `FirstHalf("abcde")` returns `"ab"`

```
static string FirstHalf(string s)
```

10. Assume you can use the function `UI.PromptInt` from class. Code the function `GetOdd`, that returns an odd number entered by the user. Keep prompting the user, "Enter an odd number: ", until the user does it. Return the odd number. Hint: an odd number is not divisible by 2. A number that is divisible by 2 has no remainder when divided by 2. Here are two example user sequences, with user responses shown **boldface.** After the left sequence, 3 is returned. After the (short) right sequence, 7 is returned:

   Enter an odd number: **4**
   Enter an odd number: **56**
   Enter an odd number: **3**

   Enter an odd number: **7**

```
static int GetOdd()
{
```

SOLUTIONS ON NEXT PAGE

Solutions to Sample Problems for Exam 1

1. output:
   a: 2 b: 13
   a: 28 b: 41

Line by line: (something like the following is encouraged to help with partial credit.)

line	a	b	comments
1.			
2.	2	13	5 + 2*(3 + 1) = 5+8 = 13
3.	2	13	print values 2 and 13
4.	28	13	2 + 2*13 = 2 + 26 = 28
5.	28	41	28 + 13 = 41
6.	28	41	print values 28 and 41

2a. `ACD` b. `BCD` c. `AD` d. `BD`

Line by line:

a. line 3: 3 < 4 true; line 4: print "A"; skip else lines 5, 6; line 7: 3+4 < 50 true; line 8: print"C"; (in no if statement!) line 9: print "D"

b. line 3: 5 < 4 false; skip line 4; line 5: 5 < 2*4 true; line 6: print "B"; line 7: 5+4 < 50 true; line 8: print"C"; line 9: print "D"

c. line 3: 30 < 40 true; line 4: print "A"; skip else lines 5, 6; line 7: 30+40 < 50 false; skip 8; line 9: print "D"

d. line 3: 50 < 40 false; skip line 4; line 5: 50 < 2*40 true; line 6: print "B"; line 7: 50+40 < 50 false; skip 8; line 9 print "D"

3. `CDEFGHIp` (final answer from last line. Steps below:)

1. n becomes 7.
2. u becomes "cdefghi" 7 characters starting from index 2
3. ch becomes 'p' count index from 0
4. print "CDEFGHI" + 'p' = “CDEFGHIp”

4. `mli` // each time through loop print next even index characters (0, 2, 4) with index < 5; all same line

Line by line:

line	i	comments
1 -		set s to `"multiply"` indices `01234567`
2	0	
3	0	0<5 is true: loop
4	0	print s[0], ‘m’, stay in same line
5	2	0+2=2
3	2	2<5 is true: loop
4	2	print s[2], ‘l’, stay in same line
5	4	2+2=4
3	4	4<5 is true: loop
4	4	print s[4], ‘i’, stay in same line
5	6	4+2=6
3	6	6<5 is false: skip loop

5. `Here are 12.5 and 15!`

Plug into locations with braces, and keep the rest of the format string!

Index 0: copy: Here

Index 1: extra formatting with :F1 for double parameter at index 1: to one decimal place, rounded

Index 2: a+b = 5+10=15.

6. Output and logic are shown for each call to TestIt. There is no advance to next line:

output line comments

2 5 E 2: 2 divisible by 2 or 5 divisible by 3 = true or false = true
3: 2 < 10 and 3 < 2 = true and false = false skip to 5,6
6: 5 < 20 true
7: print E, skip else line 8

4 5 D 2: 4 divisible by 2 or 5 divisible by 3 = true or false = true
3: 4 < 10 and 3 < 4 = true and true = true
4: print D, skip else line 5, AND skip outer else line 8

13 6 E 2: 13 divisible by 2 or 6 divisible by 3 = false or true = true
3: 13 < 10 and 3 < 13 = false and true = false skip to 5,6
6: 13< 20 true
7: print E, skip outer else line 8

12 22 2: 12 divisible by 2 or 22 divisible by 3 = true or false = true
3: 12 < 10 and 3 < 12 = false and true = false skip to 5,6
6: 22< 20 false skip 7, skip outer else line 8 (so no letter printed)

5 4 F 2: 5 divisible by 2 or 4 divisible by 3 = false or false=false, skip to outer else line 8,9
9: 5+4 >5 true
10: print F, skip else line 11

1 2 G 2: 1 divisible by 2 or 2 divisible by 3 = false or false=false, skip to outer else line 8,9
9: 1+2 >5 false, skip to else line 11, 12
12: print G

7. 28:4:10 //final answer - final values printed with ':' between them. Brief summary:
z = 2(1+4) = 10 from value returned by call to foo, passing values so a=1, b=4
x changed to 2(4+10) = 28; from value returned by call to foo, passing values so a=4, b=10
Line by line:

line x y z a b c comments
1 1 4 - - - -
2 1 4 ? first call foo, passing 1, 4
5 - - - 1 4 - set foo formal parameters
6 - - - 1 4 5 1+4=5
7 - - - 1 4 5 return 2*5 = 10
2 1 4 10 - - - back to Main scope; assign return value to z
3 1? 4 10 - - - call foo, passing 4, 10 (x not yet changed)
5 - - - 4 10 - set foo formal parameters
6 - - - 4 10 14 4+10=14
7 - - - 4 10 14 return 2*14 = 28
3 28 4 10 - - - back to Main scope; assign return value to x
4 print values with colons in between 28:4:10

8. Only one line needed in function IsLegalTriangle. A longer version with if-else is OK but redundant!

```
{
   return a < b+c && b < a+c && c < a+b;
}
```

9.

```
{
   int size = s.Length/2; // drops the remainder if the length is odd
   return s.Substring(0, size);
}
```

10. Basic interactive while loop. Continue loop on NOT odd, so continue when even;

```
{
   int n = UI.PromptInt("Enter an odd number: ");
   while (n % 2 == 0) {
      n = UI.PromptInt ("Enter an odd number: ");
   }
   return n;
}
```