The main objective of today’s lab is to solve a problem that uses complex conditional logic including relational operators, logical operators, modulus, single-alternative ifs, double-alternative ifs, and multiple-alternative ifs. You are to continue using the Visual Studio debugger to help identify any logical errors that your program contains.

1. Be sure your output looks exactly like the specified output.
2. Submit your solutions folder with the completed project to **CS150-01 Drop** folder when you are done.
3. Use the program skeleton and add comments to your code.
4. Write small pieces of code and test as you go!!!!!!!!
5. **Make certain that you don’t use any magic constants in your programs!**

**Lab 5.1**

Write a complete C++ program in a project called **05_1_Calculator** that implements a simple calculator. The operators are +, -, *, and /. Any other operator is invalid. Here is how your program is to work:

```
*** Simple Calculator ***
Select operator (+, -, *, /): *
Enter two operands: 5 3
5 * 3 = 15
Press any key to continue . . .
```

```
*** Simple Calculator ***
Select operator (+, -, *, /): ^
Enter two operands: 3 5
Error! operator is not correct
Press any key to continue . . .
```

▶ **STOP – Show the instructor or TA**
Lab 5.2

Add a project called 05_2_FunWithNumbers to your PUNetIDLabs solution that solves the following problem.

Write a complete C++ program that allows the user the ability to enter a number in the range of some lower bound (minimum 2) to some upper bound (maximum 99, inclusive). These are integers and are constants that are initially set to 2 and 99. You are then to ask the user to enter a value and print out whether the value entered is even or odd, prime or not, and finally the sum of the digits.

Here is how your program is to work:

Note that in general, when testing whether a given positive integer \( n \) is prime, it suffices to check divisibility by any prime number from 2 up to \( \sqrt{n} \) (why?). For example, to test whether \( n = 15 \) is prime, it suffices to check whether 15 is divisible by any prime numbers less than \( \sqrt{15} \approx 3.8 \). In particular, we only need to test for divisibility by 2 and 3 in this case, and since 15 is divisible by 3, then \( n = 15 \) is not prime.

Similarly, notice that to test the largest number in our set \( n = 99 \) for primality, it suffices to check for divisibility by any prime numbers less than \( \sqrt{99} \approx 9.94 \), and we know the only such numbers are 2, 3, 5, and 7. Since 99 is divisible by 3, then we know it is not prime.

The moral of these examples is that you will need to handle the cases of 2, 3, 5, and 7 separately, and for any other number between 2 and 99 \textbf{that is not one of these numbers}, to test for primality it suffices to test for divisibility by these four prime numbers. The table at http://www4.ncsu.edu/~ahjones3/courses/PrimeFactorization.pdf may be of help in clarifying this idea.

Have the instructor or TA after each step: 1) even/odd, 2) prime/not prime, 3) sum of digits.

Your programs are to compile without any errors or warnings. Once your projects are complete, place your solution PUNetIDLabs into the CS150-01 Drop folder on grace by this Friday at 5PM. Your solution is to have all previous projects completely working and correct.