Project 8 (30 points): If statements

➔ 10% bonus if your program output matches the sample solution exactly ➔

Submit Your Solution Using ANGEL by Noon on Friday, 3/1/13

(A late submission drop box will be available on 03/1/13 from Noon to 2pm)

Obtaining Project 8 Input Order and Output Requirements:

(Sample Solution and Comparison Script)

To view the order and style of the input and output information for the project, run the provided solution by **typing the following at a terminal window prompt.** (**Note**: Input files must be present in the directory that the terminal window is currently in.)

Sample Solution – for determining vertical spacing differences

/home/work/cpe112/Executables/Project_08/Project_08_solution

Values to enter for testing are: –1, 2, 3, 71, 101, 5000 and any others you want to try
Use whatever name you want for the output file. out#.txt where # represents the max numbered entered is a good idea. **Input files in P8_in.zip can be used for redirected input**

Comparison Script – for determining line differences

/home/work/cpe112data/Project_08/CompareSolution.bash Project_08.cpp

Project 8 Restrictions

Only material from Chapters 1 through 7 and any extra code in this handout is allowed.
You cannot use any C++ techniques that are covered in Chapters 8 and higher.

You are not allowed to use any global variables. Global variables are declared above the **int main()** line in a program. Global constants can be used

Starting Project 8:

- Open a terminal window and move (cd) into the Project_08 directory created in the CPE112_SPR13 directory (This is the directory structure created in project 1.) The command for this is: **cd CPE112_SPR13/Project_08 (typing cd ~/CPE112_SPR13/Project_08 works as well)**. You will need to modify the names as necessary to match your capitalization style for the two directories.

- Download all needed files from ANGEL into this directory. Using your favorite text editor (i.e. gedit), open the existing **Project_08.cpp** file (this is the header file created in Project 2) and add code to it to complete this project. Once you have finished with the program and it compiles without syntax errors, run the executable and verify that the output for your program matches the output from the provided solution executable.

- Remember to use the firefox browser when viewing ANGEL. Also, to view a pdf file saved in a directory, type the command **evince filename.pdf** at the prompt in a terminal window that has the same working directory as the directory in which the file is present.

- **Once you are satisfied with your solution, submit Project_08.cpp via Angel**.

NOTE: make sure that you do not change the order in which the information is entered. An automatic script is used to process all lab submissions, and if the order of the input information is modified, the script will not work properly with your program.

Project 8 Description

This program requires the utilization of concepts in Chapters 5, 6 and 7. This program uses two methods to determine the prime numbers that exist between 2 and a maximum value provided by the user. The prime numbers along with some other information is written to an output file specified by the user.

The program is to have an outer loop that is controlled by the maximum value entered by the user. As long as the value entered is greater than or equal to 2, the program executes and finds all primes from 2 to the number entered. For values less than 2, the program terminates. Each time through this loop, the user enters the name for an output file. **Run the sample solution to understand how the program is to operate.**

The overall construct of the program is an outer loop that is controlled by the maximum value entered by the user (use a while loop). Inside this loop, the output file is continuously prompted for and opened until it is successfully opened, and the two methods for determining the prime numbers are run. Each method requires 2 for loops (for loops are much better suited for this part of the problem than are while loops). The outer for loop runs through all the possible numbers to test, and the inner for loop performs the test on the outer loop numbers.

For each of the two methods used to determine the prime numbers, a **loop iteration counter**, that counts how many times the loops are executed, and an **event counter**, that counts the number of prime numbers found, are needed. The prime numbers are written to the output file as they are found. **Use a field width of 6 and left justification** when printing out the prime numbers. **Each time the outer loop OR the inner loop is executed, the iteration counter is to be incremented.**

Since two methods are going to be compared, each method should have its own variables for the iteration counter and event counter.

After both methods have run, output information is to consist of the number of prime numbers found, the number of iterations for each method and the ratio of method 1 iterations to method 2 iterations**. Look at the output produced by the sample solution to see what this output looks like**. **After all information has been written to the output file, close the output file stream so that it can be used with the next number to check.**

Prime Number Information

A prime number is any integer that is greater than 1 that is only divisible by itself and 1. The first few prime numbers are **2, 3, 5, 7, 11, 13, 17, 19, 23, 29, …** This program is to use two methods to determine the prime numbers between 2 and a maximum number provided by the user (You can assume that the value entered will be less than the largest integer allowed). An easy way to determine if a particular number is prime is to divide it by all integers between 1 and itself (i.e. 2, 3, … , num –1). If the number is divisible by any of these numbers, then the number in question is not prime. An easy way to perform the math is using the modulo operation. Remember that x%y (x mod y) results in a value of 0 if y divides into x.

As an example, determine if 7 is prime. 7%2 = 1, 7%3 = 1, 7%4 = 3, 7%5 = 2, 7%6 = 1. Since no number divides into 7 with a remainder of 0, 7 is a prime number.

The program is to test all numbers between 2 and the max value entered. All prime numbers found in that range are written to an output file, and this output occurs when a number is found to be prime. **The prime numbers should be printed 10 to a line. So, after printing the tenth prime number on a line, the line is terminated with a newline character.**

Method 1: Brute Force

For this method, no finesse is given to the algorithm, and all possible values are tested to determine if a number is prime or not. Two loops are required to perform the work for this method. The outer for loop control variable runs through all numbers between 2 and the maximum value entered inclusively (which means also test 2 and the maximum value). **The outer loop control variable is the number being tested to see if it is a prime number**. The inner for loop control variable runs through all the numbers that may possibly divide the outer loop control variable. It has values from 2 to the current outer loop control variable minus 1 (in other words if the maximum value entered by the user is 50, and the outer for loop control variable is currently at 25, then the inner for loop control variable is to range from 2 to 25 –1 = 24). Once the outer loop control variable has been found to be divisible by the inner loop control variable, the inner loop control variable is set to a value that makes the test expression false so that the inner loop is exited. This procedure allows for skipping any future tests on a particular number.

For example, if 25 is the value of the outer loop control variable, then the inner loop will try dividing it by all numbers between 2 and 24. If all of those divisions result in a non-zero remainder, then 25 would be declared prime. In this case, 25 is divisible by 5. When that is determined, the inner loop control variable would be changed from 5 to 26 or higher so that the loop would exit.

Hint: A Boolean variable can be used to indicate the status of the number being tested. This variable would be set to true before the inner loop is started. If a number divides the number being tested, the value would be changed to false. Then, after the inner loop a test can be performed on the Boolean variable to determine if the number in question was prime or not

Method 2: More Efficient Method

The code for this method is almost identical to that written for Method 1. Once method 1 is written and working correctly, copy it for method two. Only a few changes will have to be made to obtain the method described.

For this method some short cuts are taken. First, note that if a number (other than 2) is divisible by 2, then it is not prime. Therefore, the number being tested (the outer loop control variable) is checked to see if it is divisible by 2 provided that it is not 2. This test is performed in the outer loop, but before the inner loop is entered. If it is divisible by 2, the number is not prime, and no further testing is necessary. If it is not divisible by 2, then it only needs to be tested by odd numbers starting with 3. Therefore the inner loop control variable starts with a value of 3 that is incremented by 2 each time through the loop (test using values of 3, 5, 7, 9, …).

Second, the ending value for the inner loop control variable does not have to be number –1 where number is the number being tested. The ending value for the inner loop control variable can be number/2 + 1 (note this is integer division so 19/2 +1 gives a value of 10.

The above two changes are all that is necessary to modify method 1 into method 2. An example of this method is now given. Assume that the outer loop control variable currently has a value of 19. First, 19 is tested for divisibility by 2. It is not, so continue on to the inner loop. The starting value for the loop control variable of the inner loop is 3, and it is incremented by 2 with each loop iteration until it exceeds 19/2 + 1 = 10. Therefore, 19 is tested for divisibility by 3, 5, 7, and 9. None of these numbers divide 19, so 19 is a prime number.

Input is to follow the same order as that illustrated by the provided solution executable. Output is to be similar to that shown by the provided solution.

For comparison between the supplied solution and your solution, try maximum values of –1, 1, 2, 3, 50, 101, 5000 and any others you want to try (do not exceed 100,000).

<Project 8 Hints>

- Several header files are required: **string, iostream, iomanip, and fstream**.
- The rows of dashed lines and asterisks in the output are 60 characters long – use `string(60,’-‘);`
- **The file open error message has 11 *’s and 2 spaces on either side of the title and 41 *’s across the bottom**
- The iteration counter is incremented once for every outer loop execution and once for every inner loop execution
- The program continues to check for prime numbers until the user enters a value less than 2
- **Assume integer values only are entered- no need to verify the value input**
- Output file is continually prompted for until it is successfully opened
- **Look over the slides for additional information**
- Make sure all information – headers, rows of *’s or –‘s and prime numbers are output and that the output matches the output of the sample solution.
- Use an expression similar to numPrimes%MAX_PER_LINE to determine when a line of 10 prime numbers should be line terminated.
- The following two lines will properly line terminate a row of prime numbers after all numbers to test have been tested: This keeps a blank line from being output when the number of primes found (numPrimes in this code segment) is a multiple of MAX_PER_LINE (which is 10 for this program)

```
if (numPrimes%MAX_PER_LINE != 0)
        outFile  << endl;
```

- Information written to the output file for each method consists of a row of *’s, a title line, a row of –‘s, a list of the prime numbers (10 per line) followed by a row of *’s
- Be careful of integer division when calculating the ratio
- **Run the sample solution**

<Project 8 C++ Concepts Explained>

The following C++ concepts are included or introduced in this project:

while(expression) Statement A while loop executes the statement (the loop body) as long as the expression is true. Once the expression becomes false, the loop is exited.

Count-controlled loop: A loop that executes a specific number of times

Iteration counter: A counter variable that is incremented with each iteration of a loop.

Event counter: A variable that is incremented each time a particular event occurs.

Loop control variable: A variable used in a count-controlled loop. This variable is initialized to the starting value for a loop and then incremented (or decremented) in some fashion during the execution of the loop. It is typically used in the loop expression that is being tested.

for (InitStatement Expression1; Expression2) Statement A for loop is more compact version of a while loop. It simplifies the writing of count-controlled loops.

InitStatement: The statement (ends in a semicolon) that initializes the loop control variable.

Expression1: Same as the while condition (the expression that is tested in a while loop). Typically this is a test on the loop control variable.

Expression2: Expression that modifies the loop control variable in some manner.