week 4 Exercise

Part C: Programming Part

Instruction: Complete the following programming exercises.

1. Write a program to display all prime numbers from 1 to 100.

2. Write a program that reads an integer and then displays the number that has the same digits in the reverse order, as illustrated by this sample run:

```
This program reverses the digits in an integer.
Enter a positive integer: 123456789
The reversed integer is 987654321
```

6. A palindrome is a number or a text phrase that reads the same backwards as forwards. For example, each of the following five-digit integers are palindromes: 12321, 55555, 45554 and 11611. Write a program that reads in a five-digit integer and determines whether or not it is a palindrome. (Hint: Use the division and modulus operators to separate the number into its individual digits.)

7. A nested **for** loop is a **for** loop within another **for** loop. An example is shown below:

```
for(int i = 1; i <= 5; i++) {
      for(int j = 1; j <= 3; j++) {
            for(int k = 1; k <= 4; k++)
                  cout << '*';
            cout << endl;
      }
      cout << endl;
}
```

Try to run the code above and understand the output of the program. What does the program do? Describe your understanding by adding comments for each line.

8. Write a program that reads in the size of the side of a square and then print a hollow square of that size out of asterisks (*) and blanks. Your program should work for squares of all sizes between 1 and 20. For example, if your program reads a size of 5, it should print:

```
* * * * *
*     *
*     *
*     *
* * * * *
```

(Hint: Use nested **for** loop)

9. The factorial of a nonnegative integer n is written as $n!$ (pronounced as "n factorial") and is defined as follows:

$n! = n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 1$ (for values of n greater than or equal to 1)

or $n! = 1$ (for $n = 0$)

For example, $5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 120$. Write a program that reads a nonnegative integer and computes and prints its factorial.

10. An e-commerce website sells four different products whose retails prices are:

Product 1- $2.90
Product 2- $3.50
Product 3- $5.00
Product 4- $6.90

Write a program that let user enters the products he/she wishes to purchase and then display the total prices of purchase. Your program should use **switch** statement to help determine the retail price for each product. An example is shown below:

```
Enter product number (-1 to check-out): 1
Total purchase: $2.90
Enter product number (-1 to check-out): 2
Total purchase: $6.40
Enter product number (-1 to check-out): 4
Total purchase: $13.30
Enter product number (-1 to check-out): -1
Final purchase: $13.30
```

11. A right triangle can have sides that are all integers. The set of three integer values for the sides of a right triangle is called a Pythagorean triple. An example of Pythagorean triple is 3, 4 and 5. These three sides must satisfy the relationship that the sum of the squares of two of the sides is equal to the square of the hypotenuse, i.e. `side1`^{2} + `side2`^{2} = `hypotenuse`^{2}. Find all Pythagorean triples for `side1`, `side2` and the `hypotenuse` all no larger than 500. Use a triple-nested **`for`**-loop that tries all possibilities. This is an example of "brute force" computing. You will learn in more advanced computer science courses that there are many interesting problems where there is no known algorithmic approach other than using sheer brute force.

12. In 1979, Douglas Hofstadter, Professor of Cognitive Science at the University of Indiana, wrote *Gödel, Escher, Bach*. The book won the Pulitzer Prize for Literature and has over the years become one of the classics of computer science. Much of its charm comes from the mathematical oddities and puzzles it contains, many of which can be expressed in the form of computer programs. Of these, one of the most interesting concerns the sequence of numbers formed by repeatedly executing the following rules some positive integer *n*:
    - If *n* is equal to 1, you've reached the end of the sequence and can stop.
    - If *n* is even, divide it by two.
    - If *n* is odd, multiply it by three and add one.

    Although is also goes by several other names, this sequence is often called the *hailstone sequence* because the values tend to go up and down before coming back to 1, much as hailstones in the clouds in which they form.
    Write a program that reads in a number from the user and then generates the hailstone sequence from that point, as in the following sample run:

```
Enter a number: 15
15 is odd, so I multiply by 3 and add 1 to get 46
46 is even, so I divide it by 2 to get 23
23 is odd, so I multiply by 3 and add 1 to get 70
70 is even, so I divide it by 2 to get 35
35 is odd, so I multiply by 3 and add 1 to get 106
106 is even, so I divide it by 2 to get 53
53 is odd, so I multiply by 3 and add 1 to get 160
160 is even, so I divide it by 2 to get 80
80 is even, so I divide it by 2 to get 40
40 is even, so I divide it by 2 to get 20
20 is even, so I divide it by 2 to get 10
10 is even, so I divide it by 2 to get 5
5 is odd, so I multiply by 3 and add 1 to get 16
16 is even, so I divide it by 2 to get 8
8 is even, so I divide it by 2 to get 4
4 is even, so I divide it by 2 to get 2
2 is even, so I divide it by 2 to get 1
```

    One of the fascinating things about the hailstone sequence is that no one has yet been able to prove that the process always stops. The number of steps in the process can get very large, but somehow, it always seems to climb back down to one.