CMPT 130: Week 6 Lab Work

Most of the questions in this lab work will ask you to write a function to perform some computation. In each case, you must specify the **pre-condition and post-condition** for the function and if the function seems too complicated for you then use **supporting/helper functions** to make the problem solving easier. Last but not least, test your function(s) by writing an **appropriate test main program** and run the program several times in order to test your function with different argument values. Select the test values intelligently to test how your function behaves for some values that require special attention.

1. Write a C++ function named `successiveSum` that takes one integer argument **n** and returns the sum of the first **n** integers; that is **1+2+3+….+n.** What is the pre-condition for the function?

2. What does your `successiveSum` function defined in Question #1 return if the pre-condition for the function is violated? 0

3. Write a C++ function named `successiveProduct` that takes one integer argument **n** and returns the product of the first **n** integers; that is **1*2*3*….*n.** What is the pre-condition for the function?

4. What does your `successiveProduct` function defined in Question #3 return if the pre-condition for the function is violated? 1

5. **Even/Odd Test: -** Write a function named `isEven` that takes one integer argument and returns true if the argument is even; returns false otherwise.

6. Write a C++ function named `isDigit` that takes a character argument and returns true if the argument is a digit and returns false otherwise.

7. Write a function named `printIncreasingOrder` that takes three float arguments and prints them in increasing order. What does your function return?

8. Write a function named `reversedInteger` that takes one integer argument and returns an integer made up of the digits of the arguments in reverse order. Example,

   `reversedInteger(65)` must return 56
   `reversedInteger(0)` must return 0
   `reversedInteger(-762)` must return -267

9. A positive integer number **n** is called composite if the number is divisible by any one of the integers **2,3,4,…,n-1**. Write a C++ function named `isComposite` that takes one integer argument and returns true if the argument is composite and returns false otherwise. What is the pre-condition for the function?

10. Re-write the `isPrime` function discussed in class so that your function will not have any loop. Instead it must make use of your function `isComposite` to determine if the argument is prime or not prime.

11. Write a C++ function named `printPrimes` that takes one integer argument **n** and prints all the primes in the range [2, n].

12. Write a function named `allEven` that takes one positive integer argument **n**. Your function then must generate **n** random integers, prints each of the random numbers generated, and finally return true if all the randomly generated integers are even numbers and returns false otherwise.

13. Write a function named `allPrimes` that takes one positive integer argument **n**. Your function then must generate **n** random integers in the range [2, 100], prints each of the random numbers generated, and finally returns true if all the randomly generated integers are prime numbers and returns false otherwise.

14. Write a function named `quadraticTester` that takes three float arguments **a**, **b**, and **c** and that returns the number of real solutions (int data type) of the quadratic equation given by $\mathbf{ax^2+bx+c=0}$

15. Write a function named `allIncreasing` that takes one positive integer argument **n** and that generates n random integers, prints each of the random numbers generated, and finally returns true if the generated random integers appeared in increasing order; otherwise returns false.

16. Write a function that takes one upper case English letter character argument and that returns the English letter that is ten letters away from the argument. For example,

If the argument character is 'A' then your function must return 'K'
If the argument character is 'B' then your function must return 'L'
If the argument character is 'C' then your function must return 'M'
If the argument character is 'Q' then your function must return 'A'
If the argument character is 'V' then your function must return 'F'
If the argument character is 'Z' then your function must return 'J'

Observe that if the character that is ten letters away from the argument is outside of the upper case English letters; then we must go back to 'A'.

17. Vector Length:- Given two points on a plane P1(x1, y1) and P2(x2, y2); the length of the line segment connecting P1 and P2 is given by $\boldsymbol{d = \sqrt{(x1 - x2)^2 + (y1 - y2)^2}}$. Write a C++ function named `vectorLength` that takes four arguments x1, y1, x2, and y2 and returns the length of the line segment connecting the two points P1 and P2.

18. Area of Triangle:- Consider the area of a triangle whose three vertices on the plane are given as points P1(x1,y1), P2(x2,y2),and P3(x3,y3). Write a C++ function named `areaOfTriangle` that takes six arguments x1, y1, x2, y2, x3, and y3 and returns the area of the triangle. Assume no two points are collinear. **Hint:-** First calculate the length of each of the sides of the triangle using the Euclidean Distance formula. For this you should make use of your function `vectorLength`.

19. Simple Interest Calculator:- Write a C++ function named `accruedTotal` that that takes two float arguments: **a principal amount** and **time period in years** and returns the accrued total amount. The accrued total amount is calculated as

totalAmount = principalAmount * (1.0 + rateOfInterest * timePeriod)

For the rate of interest use the following table:

Principal Amount	Rate of Interest
[0.00, 1000.00)	2.5%
[1000.0, 10,000.00)	2.0%
[10,000.00, 100,000.00)	1.5%
≥ 100,000.00	1.0%

The main program is given below in order to test your function.

```
int main()
{
	float p, t;
	cout << "Enter principal and time period: ";
	cin >> p >> t;
	float total = accruedTotal(p, t);
	cout << "The accrued total amount is " << total << endl;
	system("Pause");
	return 0;
}
```

20. Polynomial Function:- Given the cubic polynomial function

$$f(x) = ax^3 + bx^2 + cx + d$$

Write a C++ function named f that takes five arguments **a**, **b**, **c**, **d**, and **x** where **a**, **b**, **c**, and **d** are the coefficients of the cubic polynomial and **x** is a point whose ***f(x)*** we would like to compute, and returns the value of ***f(x)***. In order to make it a general C++ function that works for any cubic order polynomial; this function must take all the coefficients **a**, **b**, **c**, and **d** as arguments. In addition, the function has to take the value of **x** as an argument. The function returns ***f(x)***.

The declaration of the function and the testing main program is given below in order to get you started.

```
float f(const float a, const float b, const float c, const float d, const float x)
{
	//Put function body here
}
int main()
{
	float a, b, c, d;
	cout << "Enter the coefficients of the cubic polynomial: ";
	cin >> a >> b >> c >> d;
	//Now let us find f(x) for ten different values of x
	float x;
	for (int i = 0; i < 10; i++)
	{
		cout << "Enter value of x ";
		cin >> x;
		cout << "The value of f(" << x << ") = " << f(a,b,c,d,x) << endl;
	}
	system("Pause");
	return 0;
}
```

21. Roots of a cubic polynomial function:- Given the cubic polynomial function

$$f(x) = ax^3 + bx^2 + cx + d$$

Write a C++ function named **polyRoot** that takes four arguments **a**, **b**, **c**, and **d** which are the coefficients of the cubic polynomial and such that $a \neq 0$ and that returns a root of ***f*** (that is, a value of ***x*** that $f(x) = 0$).

Please note that given an odd degree polynomial function (that is its highest power is odd integer) such as a cubic polynomial, it guaranteed that the graph of the function crosses the x-axis at least once which means that there exists at least one value of ***x*** such that $f(x) = 0$. Of course there may be more than one values of ***x*** such that $f(x) = 0$ in which case your function can return any one of them.

One way to compute a root of an odd degree polynomial will be as follows. First of all recall that a function ***f(x)*** has the same roots as the function $-f(x)$. With this in mind, if the coefficient $a < 0$, then multiply each

of the coefficients ***a***, ***b***, ***c***, and ***d*** by -1 so that the coefficient a > 0 and thus the function becomes an increasing function. Then perform the following steps:

Step 1. Select two initial guesses ***x1*** and ***x2*** such that ***f(x1) < 0*** and ***f(x2) > 0***. Since we made sure ***a > 0*** and thus ***f*** is an increasing function, then the choice of ***x1*** and ***x2*** can be achieved by selecting some big negative value for ***x1*** and some big positive value for ***x2***. That is to say there exists a root of f in the range **[*x1, x2*]**.

Step 2. Compute the middle value ***x = (x1+x2)/2.***

Step 3. Compute ***f(x)*** and

- If ***f(x) = 0*** then we are done we have found a required solution.
- Else if ***f(x) > 0*** then there has to be a solution in the range **[*x1, x*]**. Therefore we make the update by assigning ***x2*** the value of ***x*** and go to Step 2 above.
- Else (that is ***f(x) < 0***) then there has to be a solution in the range **[*x, x2*]**. Therefore we make the update by assigning ***x1*** the value of ***x*** and go to Step 2 above.

Repeat this process until you find a middle value x such that ***f(x) = 0***.

Remark 1. The easiest way to compute the initial guesses x1 and x2 is to initialize x1 = -1.0 and x2 = 1.0 and then decrement x1 until ***f(x1) < 0*** and increment x2 until ***f(x2) > 0***.

Remark 2. In step 3 above, it is possible f(x) is never exactly equal to zero and therefore we may end up in an infinite loop. In order to avoid infinite loop, we should stop the loop when f(x) is very close to zero such as when ***abs(f(x)) < 0.00001***. This will give us a solution correct to four decimal places which is great.

This method of computing a solution by repeatedly bisecting a given interval into two sub-intervals is known as the bisection method. It is a very easy to understand and powerful numerical method that can be used in many practical problems; albeit it is a very slow algorithm.

22. **Bisection method for computing square roots of non-negative numbers:-** Write a C++ function named **`mySquareRoot`** that takes a non-negative double data type argument a and returns the square root of a.

    Hint:- Use the bisection method discussed above with $f(x) = x^2 - a$ and the initial guesses of $x1 = 0.0$ and $x2 = a + 1$. Below is the function declaration and a test main program to test your function.

```
double mySquareRoot(const double a)
{
	//Pre-condition: a is a none negative double data type value
	//Post-condition: returns the square root of a
	//Algorithm: Bisection method
}
int main()
{
	double num;
	cout << "Enter a non-negative number ";
	cin >> num;
	double y = mySquareRoot(num);
	cout << "The square root of " << num << " is " << y << endl;
	system("Pause");
	return 0;
}
```