LAB: 4

Title : Functions

Lab Objectives: After performing this lab, the students should be able to

- Explain the concepts of functions.
- Explain what a function prototype is and how that is different from the function definition.
- Convert the code processing in the main function to a function called from the main function.

Background:

The function is a good mechanism to encapsulate code used repeatedly in a program so that it can be called from other parts of the code. A function does not use a keyword called function but instead the programmer has to define function prototype before the main (or _tmain) function and then define the function again later. A function has the following format

```
type function_name (optional parameter list)
{
   function code;
   return value;
}
```

Here types are in general the types of C variable types including int, double, char etc. The function does some processing and the calculated value is returned using the return value; instruction in the function. In the main function or the other functions calling this function_name, the value returned is used like the instruction: `calling_value = function_name (parameters);`

A function does not need to always return a value. A function not returning a value can omit the return statement and the function type is void in this case. A function can also return a pointer to a value. Pointer is covered in later labs. A function can also return a user defined type. Function prototype has the following format:

```
type function_name (list of variable types);
```

Examples are:

```
int compute(int);
void tryout();
```

Function prototypes differ from the function definitions in two places: there is no code (no {} with code in between) and the variable names do not follow the types. A function prototype may return nothing, in which case void is the type returned; also it may have no parameters at all like the second example above.

The function prototype is often declared before the main function with the function calls inside the main function or the other functions calling this function. The function definitions are put after the main function.

C++ Standard Library: is a collection of functions, which are written in the core language and part of the C++ ISO Standard itself to provides several generic functions that utilize and manipulate these containers, function objects, generic strings and streams (including interactive and file I/O), support for some language features, and everyday functions for tasks such as finding the square root of a number. Using sqrt in <math.h> header file.

The reader can refresh the concepts of function, by referring to reference : Deitel and Deitel, How to program C++, chapter 3.

Consider the following program that computes the sum of 1 + 2 + 3 +... + 100 using a ***for*** loop.

```
#include <iostream>
using namespace std;

int main ( ) {

   int sum = 0;

   for (int j = 1; j <= 100; j++)
      sum += j;
   cout << “The sum of 1 + 2 + 3 + ... + 100 is “ << sum << endl;
   return 0;
}
```

This program shows how to compute a sum in a quick way. The program can be easily modified to compute and display the sum of 1 + 2 + … + n for a variable n (with n = 100 as above). Such code of (3 lines) computing the sum can be coded as one instruction in the main function `compute_sum(n)` with the variable `n` as the parameter.

```
#include <iostream>
using namespace std;

int main ( ) {
  cout<<”The sum of 1 + ... + 100 is”<<compute_sum(100) <<endl;
  return 0;
}

int compute_sum(int n)
{
  int sum = 0;

  for (int j = 1; j <= n; j++)
      sum += j;

  return sum;
}
```

Pre-lab

- Review the concepts of functions in Deitel Chapter 3.
- Work on the following simple examples of functions
  - Write a *function* double **quadratic** (double x) that computes function f(x) with x as the only input parameter and returns f(x) = $ax^2 + bx + c$ with a = 1.0, b = 2.0, and c = 1.0 (Use *lab3a.cpp* to test your solution)
  - Write a function **quadratic** *(double a, double b, double c, double x)* that returns f(x) = $ax^2 + bx + c$ with all a, b, c, and x, as parameters .(Use *lab3b.cpp* to test your solution)
  - Write a function distance2 that calculates the distance between two points (x1, y1) and (x2, y2) in the plane. Put all numbers and return types in double. Use (3.0, 0.0) and (0.0, 4.0) as one pair of input to verify your program. (Use *lab3c.cpp* to test your solution)

Lab Assignments:

1. Calculate Primes in a Range

In lab 1, you have written a program to input an integer n and then check if that integer n is a prime. Using a function that checks if an integer n is a prime (returns true if the integer is prime and false otherwise) and a for loop, you can check and print out all primes in a range, say for example all prime numbers less than 100, all prime numbers less than 1000, or all primes between 1500 and 1600.

Scenario

Input	Prime Numbers	#Prime Numbers
10	2 3 5 7	4
25	2 3 5 7 11 13 17 19 23	9
37	2 3 5 7 11 13 17 19 23 29 31 37	12
50	2 3 5 7 11 13 17 19 23 29 31 37 41 43 47	15
70	2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67	19
99	2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97	25

Instructions:

- Use the following template code (file: *lab3d.cpp*), Write a C program that prompts for an integer n and compute the primes less than n. Write the code as follows:

```
for(int j = 2; j <= n; j ++)
   if (isPrime (j)) // call the function isPrime (j) to check
         cout≪j ≪" is a prime"≪endl;
   else
         count≪j ≪" is not a prime"≪endl;
```

- Write the function `int isPrime (int n)` that takes an `int` type input parameter n and calculate and return the one or zero depending on whether n is a prime or not.
- Test your code with n = 20 to compute and display all integers up to 20 as a prime number.
- Modify and enhance your code to display all prime numbers from 2 to n, 8 in a row (you do not display "j is a prime" in a line, but display 2 3 5 7 11 13 17 19 in the first row, 23 29 31 37 41 43 47 53 in the second row etc...

- Enhance your program to print out the total number of prime numbers up to n. For example, the total number of primes up to 100 is 25.
- Modify your program such that, for bigger n (i.e., n > 200), you have the option to either display only the total number of primes less than n or all the prime numbers less than n (8 in a row) or both (Note: three options).

2. Maximum of three numbers

Write a program to read three integers in the main and then send them to an ***inline*** function `max` which will return the maximum number among them.

3. Fibonacci number

The *Fibonacci numbers* or *Fibonacci series* or *Fibonacci sequence* are the numbers in the following integer sequence:

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144 …

or (often, in modern usage):

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144 …

By definition, the first two numbers in the Fibonacci sequence are 1 and 1, or 0 and 1, depending on the chosen starting point of the sequence, and each subsequent number is the sum of the previous two. In mathematical terms, the sequence Fn of Fibonacci numbers is defined by the recurrence relation

$$F_n = F_{n-1} + F_{n-2},$$

with seed values

$$F_1 = 1,\ F_2 = 1.$$

or

$$F_0 = 0,\ F_1 = 1.$$

The Fibonacci sequence is named after Fibonacci. His 1202 book Liber Abaci introduced the sequence to Western European mathematics, although the sequence had been described earlier in Indian mathematics. By modern convention, the sequence begins either with $F_0 = 0$ or with $F_1 = 1$. The Liber Abaci began the sequence with $F_1 = 1$, without an initial 0.

Fibonacci numbers are closely related to Lucas numbers in that they are a complementary pair of Lucas sequences. They are intimately connected with the golden ratio; for example, the closest rational approximations to the ratio are 2/1, 3/2, 5/3, 8/5, ... Applications include computer algorithms such as the Fibonacci search technique and the Fibonacci heap data structure, and graphs called Fibonacci cubes used for interconnecting parallel and distributed systems. They also appear in biological settings, such as branching in trees, phyllotaxis (the arrangement of leaves on a stem), the fruit sprouts of a pineapple, the flowering of artichoke, an uncurling fern and the arrangement of a pine cone.

Write a function `fibonacci(n)` that will accept a number n and returns the nth number in Fibonacci sequence. Use the template code in *lab4f.cpp* to test and submit your solution.

Scenario

n	`fibonacci(n)`
0	1
1	1
2	2
3	3
4	5
5	8
6	13
7	21
8	34
9	55
10	89
11	144
12	233
13	377

Modifications

Once you managed to complete the above assignment

- Modify the main function to print the entire sequence upto n rather than just the nth number in the Fibonacci series. i.e., if the input value is 8 the output of your program should be as follows rather than being just 34

  1, 1, 2, 3, 5, 8, 13, 21, 34

- Instead of modifying the main function in as we did previously, modify the Fibonacci function to achieve the same result as the earlier modification.

4. Summation of integer digits

Write a recursive function `condense(int)` that receives an integer consisting of any number of digits. Your function should calculate and return the summation of the integer digits. For example if given the number 237834 your function should return 27 after calculating this number as

$$2 + 3 + 7 + 8 + 3 + 4 = 27$$

Use the template code in *lab4g.cpp* to test and submit your solution.

Scenario

n	`condense(n)`
237834	27
892364	32
8465	23
9489204	36
23	5
4	4

5. Factorial

The factorial of a non-negative integer n, denoted by n!, is the product of all positive integers less than or equal to n. For example,

5! = 5 x 4 x 3 x 2 x 1 = 120

The value of 0! is 1, according to the convention for an empty product. The factorial operation is encountered in many areas of mathematics, notably in combinatorics, algebra, and mathematical analysis. Its most basic occurrence is the fact that there are n! ways to arrange n distinct objects into a sequence (i.e., permutations of the set of objects). This fact was known at least as early as the 12th century, to Indian scholars. The notation n! was introduced by Christian Kramp in 1808.

Write a function `factorial(n)` that will accept a number n and returns factorial of that number (i.e., n!). Use the template code in *lab4h.cpp* to test and submit your solution.

Scenario

n	`factorial(n)`
0	1
1	1
2	2
3	6
4	24
5	120
6	720
7	5040
8	40320
9	362880
10	3628800
11	39916800
12	479001600
13	6227020800

6. Sum of even numbers in range

Write a function called `sumNums` which takes as input two integers N and M entered by the user from keyboard , and returns to the main function the summation of even numbers from N to M. (note: N and M included if were even numbers). Use the template code in *lab4i.cpp* to test and submit your solution.

Scenario

n	M	`sumNums` (n,m)
10	45	486
12	54	726
19	23	42
35	50	344
45	8	0
1447	9453	21816350
234	2945	2154684
1	1	0
2	2	2

Modifications

Once you managed to complete the above assignment

- Modify the program so that the `sumNums` function returns the sum of odd numbers from N to M rather than even numbers
- Now, Modify the program so that the main function, in addition to the tow numbers n and m, request the user and accept wither the user want to add the even or odd number. Then after the main function should pass this choice to the `sumNums` function as a third argument. Modify `sumNums` function to accept this third parameter.

7. Check Case

Write a function called `isCapital` that has one parameter and receives a character and returns 1 if the character received is a capital letter, and 0 otherwise. Then in the main function ask the user to input a character and output C if the character is in capital case and c otherwise after consulting the `isCapital` function. Use the template code in *lab4j.cpp* to test and submit your solution.