# Lab 4

Due 3/16 11:59 AM

1. Define three functions to a) get input, b) calculate its mean and c) standard deviation respectively. The following is the main function you must use and cannot change:

```
int main() {
    cout << "Your array length: ";
    int n;
    cin >> n;

    int ar[n];
    getInput(ar, n);

    cout << "mean: " << average(ar, n) << endl;
    cout << "Std: " << standardDev(ar, n);

    return 0;
}
```

Example 1:

```
Your array length: 5
Give array elements: 1 2 3 4 5
mean: 3
Std: 1.41421
```

Example 2:

```
Your array length: 7
Give array elements: 1 2 3 3 3 8 9
mean: 4.14286
Std: 2.84999
```

2. Similar to the previous problem, define three functions to a) get input, b) calculate its mean and c) standard deviation respectively. This time we use vectors. The following is the main function you must use and cannot change:

```
int main() {
    cout << "Your vector length: ";
    int n;
    cin >> n;

    vector<int> vec;
    getInput(vec, n);

    cout << "mean: " << average(vec) << endl;
    cout << "Std: " << standardDev(vec);

    return 0;
}
```

Example 1:

```
Your vector length: 5
Give vector elements: 1 2 3 4 5
mean: 3
Std: 1.41421
```

Example 2:

```
Your vector length: 7
Give vector elements: 1 3 5 7 9 11 13
mean: 7
Std: 4
```

3. Define functions with the name `overload` which helps you understand the concept of function overloading. Use the main function below:

```
int main() {
    overload();
    int i = 2;
    overload(i);
    string s = "abcde";
    overload(s);
    overload("fghij");
}
```

The output should be as follows:

```
no arg version!
overload: i++ = 3
overload(s): abcde
overload(s): fghij
```

4. Given an array `arr` of integers, a **lucky integer** is an integer that has a frequency in the array equal to its value. Define a function which returns the largest lucky integer in the array. If there is no lucky integer, return -1.

Example 1:

```
Input:
    n: 4
  arr: 2 2 3 4
Output: 2
```

Example 2:

```
Input:
    n: 6
  arr: 1 2 2 3 3 3
Output: 3
```

Example 3:

```
Input:
    n: 5
  arr: 2 2 2 3 3
Output: -1
```

5. Given the array `arr` consisting of `2n` elements in the form `[x1, x2, …, xn, y1, y2, …, yn]`. Define a function which returns the array in the form `[x1, y1, x2, y2, …, xn, yn]`.

   Example 1:

```
Input:
    n: 3
  arr: 2 5 1 3 4 7
Output: [2, 3, 5, 4, 1, 7]
```

   Explanation: Since x1=2, x2=5, x3=1, y1=3, y2=4, y3=7 then the answer is [2, 3, 5, 4, 1, 7].

   Example 2:

```
Input:
    n: 4
  arr: 1 2 3 4 4 3 2 1
Output: [1, 4, 2, 3, 3, 2, 4, 1]
```

6. Write a recursive C++ program to calculate the sum of all integers from 1 up to a given number N. The program should use recursion to achieve this, without relying on loops or arithmetic formulas for summing the series.

   Example 1:

```
Input:
    n: 15
Output: 120
```

Example 2:

```
Input:
    n: 20
Output: 210
```

7. Define a function to find the least common multiple (最小公倍數) of the given two integers.

   [Hint: It is recommended to use a recursive function to find the greatest common divisor (最大公因數) first.]

   Example 1:

```
Input: (a, b): 12 18
Output: 36
```

8. Given an integer `n`, define a recursive function which returns `true` if it is a power of three. Otherwise, return `false`. An integer `n` is a power of three, if there exists an integer `x` such that `n`==$3^x$.

   Example 1:

```
Input: n = 27
Output: true
```

   Example 2:

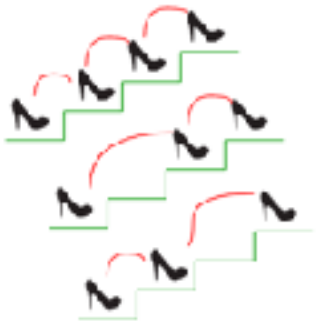
```
Input: n = 6
Output: false
```

   Example 3:

```
Input: n = 1
Output: true
```

9. You are climbing a staircase. It takes `n` steps to reach the top. Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top? Use the timing code given at

the end of this problem to help you see your program's efficiency. [Hint: You can start with a recursive function which calculates the nth Fibonacci number ($F_n = F_{n-1} + F_{n-2}$) and think of the relation between Fibonacci sequence and this problem.]



Example 1:

```
Input: n = 2
Running time: 3 microseconds
Output: 2
```

Example 2:

```
Input: n = 3
Running time: 3 microseconds
Output: 3
```

Example 3:

```
Input: n = 20
Running time: 3 microseconds
Output: 10946
```

Example 4:

```
Input: n = 30
Running time: 4 microseconds
Output: 1346269
```

Example 5:

```
Input: n = 40
Running time: 5 microseconds
Output: 165580141
```

If your answer for Example runs 3, 4 & 5 is much longer than 5 microseconds on onlineGDB, it means you have a lot of duplicated calculations in your recursion. Why?

## Timing your program

For problems related to recursion (such as Problems 6 to 8), you can use the following code (next page) to check the running time of your code since recursive functions can be extremely inefficient.

```
#include <iostream>
#include <chrono>

using namespace std;

int main() {
    auto start_time = chrono::high_resolution_clock::now();
    // code to measure the running time

    // --------------------------------
    auto end_time = chrono::high_resolution_clock::now();
    auto duration = chrono::duration_cast<chrono::microseconds>(end_time -
start_time).count();
    cout << "Running time: " << duration << " microseconds" << endl;
       return 0;
}
```