# Description

The purpose of this lab is to check your understanding of the content in lecture 4a, Arrays and Pointers.

We will focus on the material from chapter 7 of the book. The objectives are:

# To use the array data structure to represent a set of related data items.

# To use arrays to store, sort and search lists and tables of values.

# To declare arrays, initialize arrays and refer to the individual elements of arrays.

# To pass arrays to functions.

# Basic searching and sorting techniques.

# To declare and manipulate multidimensional arrays.

# To use C++ Standard Library class template vector.

# Part 1: Questions

Complete the assignment in the module *Topic 4 Lab: Arrays and Pointers* named, "Lab 4a Questions", in Canvas.

# Part 2: Activity

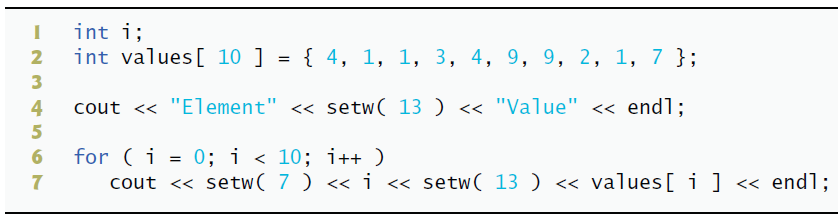
This part of the lab should be submitted using the Lab 4a Activity assignment in the *Topic 4a Lab: Arrays and Pointers* module.

**You should complete the non-programming parts of this lab using this document.** **It is possible that some of the images and answer-boxes below might move or need to be resized while using them. Do your best to make your final document neat and organized.**

## Problem 1: Program Output

For each of the given program segments, read the code and write the output in the space provided below each program. [Note: Do not execute these programs on a computer.]

1. What is output by the following program segment?



Answer:

Element Value

0 4

1 1

2 1

3 3

4 4

5 9

6 9

7 2

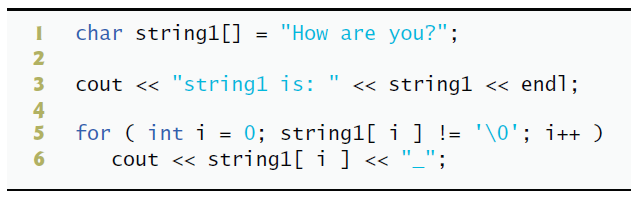
8 1

9 7

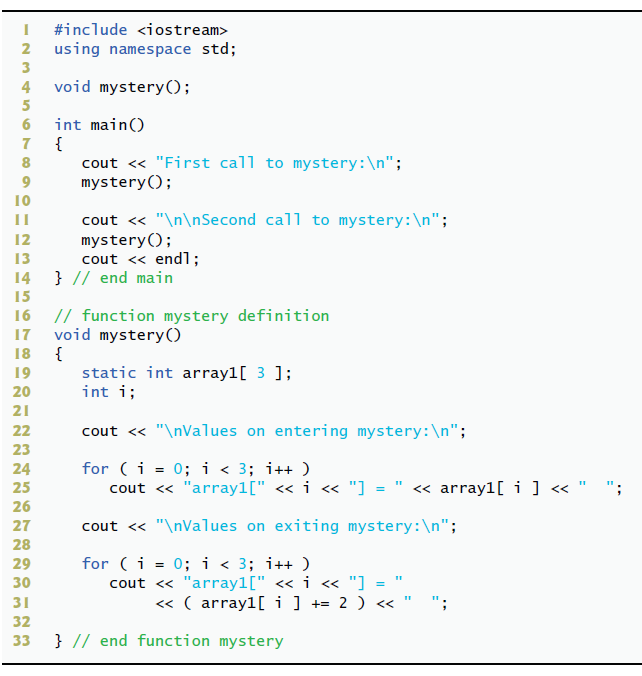
1. What is output by the following code segment?

Answer: string1 is: How are you?

H\_o\_w\_ \_a\_r\_e\_ \_y\_o\_u\_?\_



1. What is output by the following program?



Answer: First call to mystery:

Values on entering mystery:

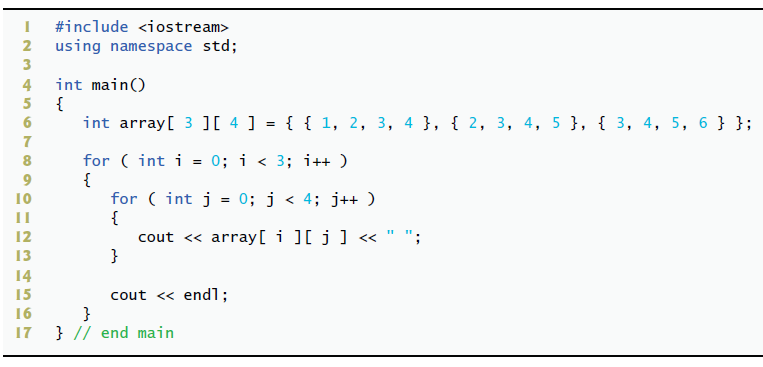
array1[0] = 2 array1[1] = 4 array1[2] = 6

Second call to mystery:

Values on entering mystery:

array1[0] = 8 array1[1] = 10 array1[2] = 12

1. What is output by the following program?



Answer: 1 2 3 4 2 3 4 5 3 4 5 6

### Problem 2: Correct the Code

For each of the given program segments, determine if there is an error in the code. If there is an error, specify whether it is a logic or compilation error, circle the error in the program, and write the corrected code in the space provided after each problem. If the code does not contain an error, write “no error.” [*Note:* It is possible that a program segment may contain multiple errors.]

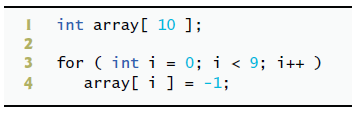
1. The following code should assign 8 to the fifth element in *array*:

problem 2.1

Answer: Logic error, 8 would be the first element in the array in this case. If we want 8 to be the 5th element then we should rewrite it like so:

array[ 5 ] = [ 0, 0, 0, 0, 8 ];

1. The for loop should initialize all array values to -1.



Answer: Logic error, this is initializing the first 9 values in the array as -1. To correct this we must change the conditions within the for loop.

int array[10];

for (int i = 0; i <= 9; i++)

array[i] = -1;

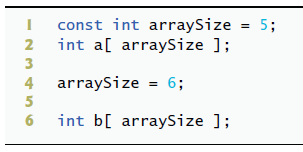
1. Array *array* should contain all the integers from 0 through 10, inclusive.

Answer: Compilation error, this array is initialized with 10 elements, but to contain all the integers from 0 to 10, inclusive we need 11 elements.

int array[11] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

problem 2.3

1. The following code segment should declare two arrays containing five and six elements, respectively:



Answer: Compilation error, we cannot declare arraySize as a constant variable and attempt to change it later. The best solution is to use one constant variable for each array.

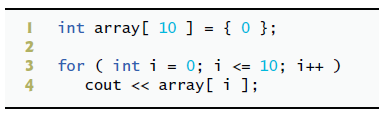
const int array1Size = 5;

int a[array1Size];

const int array2Size = 6;

int b[array2Size];

1. The for loop that follows should print *array*’s values:



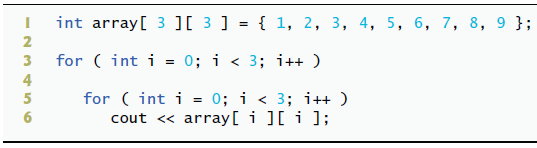
Answer: Logic error, this code actually prints values outside the bounds of the array due to the conditions of the for loop.

int array[10] = {0};

for (int i = 0; i < 10; i++)

std::cout << array[i];

1. The for loop that follows should print all of array’s values:



Answer: Logic error, for the nested for loop we should change the variable i into another variable with a different name.

int array[3][3]={1, 2, 3, 4, 5, 6, 7, 8, 9};

for(int i = 0; i < 3; i++)

for(int j = 0; j < 3; j++)

std::cout << array[i][j];

1. Research and answer the question, what are the benefits of using the *std::array*? What are the downsides?

Answer: The benefits of using std::array are that that std::arrays have value semantics, and therefore can be passed by reference or value in function arguments, and can also be returned by value. Normal C style arrays do not have value semantics and therefore if we want to pass them through a function we must also include its size.

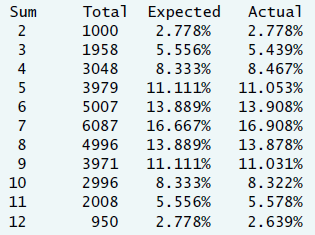
These value semantics also allow us to use C++ Standard Library’s sort function to arrange the elements in the array into ascending or descending order, which allows us to then use binary search the array for specific values.

Std::array also allows us to use ranged-base for statements to iterate through each element of the array or modify it without having to worry about exceeding the bounds of the array, as it does the bounds-checking for us.

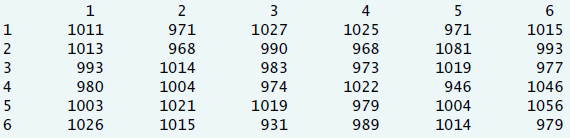
Apart from the different and perhaps awkward syntax of using std::arrays there are no inherent performance downsides to using std::arrays.

### Problem 3: Programming

1. Write a program that simulates the rolling of two dice.
   * **Call the program, "dice.cpp".**
   * The program should call rand to roll the first die, and should call rand again to roll the second die.
   * The sum of the two values should then be calculated. [Note: Each die has an integer value from 1 to 6, so the sum of the two values will vary from 2 to 12, with 7 being the most frequent sum and 2 and 12 being the least frequent sums.]
   * There 36 possible combinations of the two dice.
   * Your program should roll the two dice 600,000 times.
   * Use a one-dimensional **std::array** to tally the numbers of times each sum appears.
   * Print the results in a tabular format. **Follow the sample output exactly.**
   * Also, determine if the totals are reasonable (i.e., there are six ways to roll a 7, so approximately one sixth of all the rolls should be 7).
   * Use the *at* member function of the array class to access the elements of the array.
   * **Be sure to use constant variables for the dimension of the array.**

 Sample Output:

1. Modify the program to use a two-dimensional **std::array** and call it, **"dice\_2d.cpp"**.
   * Rather than counting the number of times each sum appears, increment the correct cell in the array that represents the first die's value (the row) and the second die's value (the column).
   * **Be sure to use constant variable(s) for the dimensions of the array.**
   * Print this array with the number of times each dice combination occurred. **The output must look like the following**:



# What to Submit for Lab 4a Activity

* A PDF of this document completed.
* dice.cpp
* dice\_2d.cpp
* A screenshot of the **output for *each program***.

Note: your screenshots must include the entire Visual Studio Code window. Do not include your desktop or anything else in the image. Do not take a picture with a camera or phone, use your computer to create a screenshot.