CSCI 1300 CS1: Starting Computing

Naidu/Correll/Yeh/Hoefer - Fall 2021

Recitation 7 - the week of October 4th, 2021

# Arrays

Last week in the lecture, we learned about the basics of arrays, their operations, and common algorithms. Today, let’s solve some problems related to arrays.

# Exercises

1. Spot the errors

a)

#include <iostream>

#include <string>

using namespace std;

int main()

{

double scores[5] = {85.4, 90.3, 100, 89, 74.5, 95};

double sum = 0;

for(int i = 0; i < 5; i++){

sum += scores[i];

}

double avg = sum / 5.0;

cout << “Average = “ << avg << endl;

return 0;

}

b)

#include <iostream>

using namespace std;

int main()

{

int temps[N] = {50, 53, 49};

cout << “The most recently recorded temperature of the day is “ << temps[3] << “ degrees Fahrenheit.” << endl;

return 0;

}

c)

#include <iostream>

using namespace std;

int main()

{

int N = 5;

int colors[] = {“red”, “blue”, “yellow”, “green”};

//printing all the colors

for (int i = 0; i < N i++){

cout << “My favorite color is “ << colors[N] << endl

}

return 0;

}

d)

#include <iostream>

using namespace std;

int main()

{

numbers[] = {};

for(i = 0; i < 10; i--){  
 i[numbers] = i + 1;

cout << numbers[] << endl;

}

return 0;

}

1. Reverse an array! There are two subproblems here:
   1. Write a function called **reverseArray()** which takes two parameters: an int array and the length of that array. The function should not return anything as arrays are passed by reference and the values will get changed in place.
   2. Write a function called **swap()** which takes three parameters: an array, startIndex, and endIndex. This function will swap the values present at the startIndex and endIndex in the given array.
   3. Write a main to initially take in the size of the array, and then instantiate and populate an array with integers by reading the array elements from the user. After creating this array, the main should then call **reverseArray()** and print the reversed array.

RULE: Do **NOT** use a temporary array. However, you can create just a temporary variable (The reversing operation should be done “in-place”)

Here is a function prototype for **reverseArray**:

**// this function is to reverse all the elements of an array**

**void reverseArray (int myArray[], int myArrayLength);**

Here is a function prototype for **swap**:

**// this function is to swap two elements in an array**

**void swap(int myArray[], int startIndex, int endIndex);**

Example output (**bold** is user input)

| Please enter the size of the array:  **10**  Please enter the elements of the array:  **1 2 3 4 5 6 7 8 9 10** The reversed array is: 10 9 8 7 6 5 4 3 2 1 |
| --- |

**2a.** Write an algorithm in pseudocode for the program above.

| **Pseudocode:** |
| --- |

**2b.** Let’s draw a flowchart of the solution

| **Flowchart:** |
| --- |

**2c.** Imagine how a sample run of your program would look like. Think about at least two examples

| **Sample Run 1:** |
| --- |

| **Sample Run 2:** |
| --- |

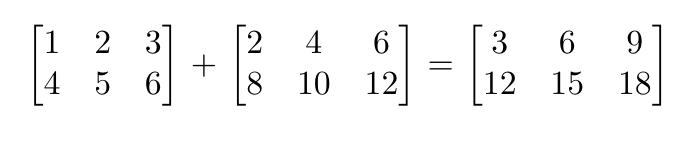
| **Sample Run 3:** |
| --- |

**2d.** Identify the values that you must test for. We call these values “**boundary conditions**”.

| **Answer:** |
| --- |

**2e.** Implement your solution in C++ using VS Code. Revise your solution, save, compile and run it again. Are you getting the expected result and output? Keep revising until you do. Make sure you test for the values used in your sample runs, and for the boundary conditions.

1. Matrix Sum



The sum of two matrices is found by adding together corresponding entries in each matrix, as shown above. (For example, the numbers at (row 1, column 1) of each matrix are added together to get the (row 1, column 1) number of the sum matrix.)

Create a function that takes as parameters two 2 x 3 int arrays, and prints their sum, displayed in two rows.

Example header:

void matrixSum(int a[2][3], int b[2][3])

Then, create a main() function that asks the user to input the values for each matrix, one row at a time. Store these in two arrays, and pass them to matrixSum.

Example output (**bold** is user input)

| Enter values for matrix 1, one row at a time:  **1 2 3**  **4 5 6**  Enter values for matrix 2, one row at a time:  **2 4 6**  **8 10 12**  The sum is:  3 6 9  12 15 18 |
| --- |

**3a.** Write an algorithm in pseudocode for the program above.

| **Pseudocode:** |
| --- |

**3b.** Let’s draw a flowchart of the solution

| **Flowchart:** |
| --- |

**3c.** Imagine how a sample run of your program would look like. Think about at least two examples

| **Sample Run 1:** |
| --- |

| **Sample Run 2:** |
| --- |

| **Sample Run 3:** |
| --- |

**3d.** Identify the values that you must test for. We call these values “**boundary conditions**”.

| **Answer:** |
| --- |

**3e.** Implement your solution in C++ using VS Code. Revise your solution, save, compile and run it again. Are you getting the expected result and output? Keep revising until you do. Make sure you test for the values used in your sample runs, and for the boundary conditions.