

CH 7: Arrays

P1. Write a program that calculates your COSC 1550 Final Grade.

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

int main()
{
    int d1,
        d2,
        d3,
        d4;
    wd1,
    wd2,
    wd3,
    wd4;
    float grade;
    cout << "Enter your scores for Program Assignments, "
        << "Chapter Tests, inal Exam, and Class "
        << "Participation, respectively" << endl;

    cin >> d1 >> d2 >> d3 >> d4;
    cout << "Enter your weights for Program Assignments, "
        << "Chapter Tests, Final Exam, and Class Participation, "
        << "respectively" << endl;
    cin >> wd1 >> wd2 >> wd3 >> wd4;

    float grade = static_cast<float> (wd1 * d1 + wd2 * d2 +
    wd3 * d3 + wd4 * d4) / (wd1 + wd2 + wd3 + wd4);

    cout << endl << "Your overall grade is: " << fixed <<
    showpoint << setprecision(2) << grade << endl;
    return 0;
}
```

VS

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

int main()
{
    const int SIZE = 4;
    int deliv[SIZE],
        wghtD[SIZE],
        wghtedSum = 0, totWghts = 0;
    float grade;

    cout << "Enter your scores for Program Assignments, "
        << "Chapter Tests, inal Exam, and Class "
        << "Participation, respectively" << endl;

    for (int i = 0; i < SIZE; i++)
        cin >> deliv[i];

    cout << "Enter your weights for Program Assignments, "
        << "Chapter Tests, inal Exam, and Class "
        << "Participation, respectively" << endl;

    for (int i = 0; i < SIZE; i++)
        cin >> wghtD[i];

    for (int i = 0; i < SIZE; i++)
    {
        wghtedSum += deliv[i] * wghtD[i];
        totWghts += wghtD[i];
    }

    grade = static_cast<float> (wghtedSum) / totWghts;
    cout << endl << "Your overall grade is: " << fixed
        << showpoint << setprecision(2) << grade << endl;
    return 0;
}
```

Consider the changes that you would need to do if the number of deliverables is changed to 10.

A1: need to add 6 more variables of d5, d6, ... and weights

A2: change the value of SIZE to 10.

		d1			
d2					
			d3		
					d4

Variables hold only one value at a time

```
int d1,  
    d2,  
    d3,  
    d4;  
char grade;
```


Allows to store and work with multiple values of the same data type. (note: the values are stored together in consecutive memory locations)

```
int deliv[4];
```

Array definition (a definition of a consecutive memory locations)

all locations have the same name (but different index).

To define an array, you should know:

1. The type of the values.
2. Name of the array
3. How many elements (aka: size declarator): a constant integer expression with a value greater than zero.

Fill the following table

		type	name	Number of elements	definition
1	empNums, a 100-element array of ints				
2	payRates, a 25-element array of floats				
3	miles, a 14-element array of longs				
4	cityNames, 126-element array of string objects				

Size declarator should be a constant integer expression with a value greater than zero.

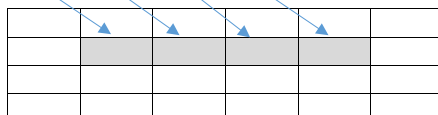
	Validity	Reason?
1. <code>int scores[10];</code>	Valid/invalid	
2. <code>int size; size = 10; int scores[size];</code>	Valid/invalid	
3. <code>int scores[-2];</code>	Valid/invalid	
4. <code>const int SIZE = 10; int scores[SIZE];</code>	Valid/invalid	

Amount of memory used by an array: (number of elements) * (number of bytes for each element)

Array Definition	Number of Elements	Size of each element	Size of the array
<code>char letters[25];</code>			
<code>short rings[100];</code>			
<code>int miles[84];</code>			
<code>float temp[12];</code>			
<code>double distance[1000];</code>			

Accessing array elements: The individual elements of an array are assigned unique subscripts.

`deliv[0] deliv[1] deliv[2] deliv[3]`



`const int SIZE = 4;`

`int deliv[SIZE]; //array definition`

assume the following numbers

90	45	60	77
----	----	----	----

`cout << deliv[0] << endl;` note: subscript is zero.

`cout << deliv[1] << endl;` note: subscript is _____.

cout << deliv[2] << endl; note: subscript is _____.

cout << deliv[3] << endl; note: subscript is _____.

cout << deliv[4] << endl; note: subscript is _____.

cout << deliv[5] << endl; note: subscript is _____.

Output the last array element

Array Definition	First array element	last array element
char letters[25];		
short rings[100];		
int miles[84];		
float temp[12];		
double distance[1000];		

Accessing array elements

int tests[4];

Assume the following initial values.

65	87	34	89
----	----	----	----

1. Elements can be accessed with a constant or literal subscript:

```
cout << tests[3] << endl;
```

output:

2. Elements can be accessed with integer expression as subscript:

```
int i = 5;
cout << tests[i] << endl;
```

output:

Array elements can be used as regular variables:

int tests[4];

Assume the following initial values.

65	87	34	89
----	----	----	----

1. `tests[0] = 79;`

Show the array elements after the above assignment statement.

--	--	--	--

2. `cout << tests[0];`

output:

3. `cin >> tests[1];`

Assume the user input was 75. Show the array elements after the above assignment statement.

--	--	--	--

4. `tests[4] = tests[0] + tests[1];`

Show the array elements after the above statement.

--	--	--	--

5. `cout << tests;`

note: Arrays must be accessed via individual elements:

output:

Using a Loop to Step through an Array

1. Fill the array according to the following code segment.

2. Write a code to define an array, numbers, and assigns 99 to each element:

3. Write a code that outputs an array which contains whole numbers on to the console:

```
const int ARRAY_SIZE = 5;  
int numbers[ARRAY_SIZE];
```

Array Initialization

Arrays can be initialized with:

1. **an initialization list.**

```
const int SIZE = 5;  
int tests[SIZE] = {79, 82, 91, 77, 84};
```

79	82	91	77	84
----	----	----	----	----

- The values are stored in the array in the order in which they appear in the list.
- The initialization list cannot exceed the array size.

Formatted: Font: Bold

2. **Partial Array Initialization**

```
const int SIZE = 5;  
int tests[SIZE] = {79, 82, 91};
```

79	82	91	0	0
----	----	----	---	---

- If array is initialized with fewer initial values than the size declarator, the remaining elements will be set to 0.
- `int tests[SIZE] = {79, , 91};` //illegal. If you leave an element uninitialized, you must leave all the elements that follow it uninitialized as well.

Formatted: Font: Bold

3. **Implicit Array Sizing**

```
int tests[] = {79, 82, 91, 77};
```

79	82	91	77
----	----	----	----

- Can determine array size by the size of the initialization list:

Formatted: Font: Bold

Overall conclusion: Must use either array size declarator or initialization list at array definition

1. What will the following code display?

```
int numbers[] = {99, 87, 66, 55, 101};  
for (int i = 1; i < 4; i++)  
    cout << numbers[i] << endl;
```

--	--	--	--	--	--	--	--	--	--

2. What will the following code display?

```
int numbers[4] = {99, 87};  
cout << numbers[3] << endl;
```

--	--	--	--	--	--	--	--	--	--

3. What will the following code do?

```
const int SIZE = 5;  
double x[SIZE];  
for(int i = 2; i <= SIZE; i++)  
{  
    x[i] = 0.0;  
}
```

--	--	--	--	--	--	--	--	--	--

35) What will the following code do?

```
const int SIZE = 5;  
double x[SIZE] = {0.0};
```

--	--	--	--	--	--	--	--	--	--

4. What will the following code display?

```
int numbers[] = {99, 87, 66, 55, 101};  
for (int i = 1; i < 4; i++)  
{  
    cout << ++numbers[i] << endl;  
}
```

--	--	--	--	--	--	--	--	--	--

5. What will the following code display?

```
int numbers[] = {99, 87, 66, 55, 101};  
for (int i = 1; i < 4; i++)  
{  
    cout << numbers[++i] << endl;  
}
```


1. Write an initialization statement to initialize an array with all zeros.

2. Write the array elements:

`int a [10] = {3, 4, 6, 2, 3};`

--	--	--	--	--	--	--	--	--	--

`int a [10] = {0};`

--	--	--	--	--	--	--	--	--	--

`int a [10] = {1};`

--	--	--	--	--	--	--	--	--	--

1. Write a program that display the first 18 array elements on the screen

2. Write a program that outputs displays the first 15 elements in the reverse order of their entry.

Functions with arrays

1. There is no such thing as pass-by-value with arrays.
2. Arrays are by definition pass-by-reference; When an array is an argument to a function, it passes the reference, i.e., the address of the first element. DO NOT PUT THE &.

3. The parameters show that the parameter is an array by putting the square brackets.

```
void readNumbers(int numbers[], int & used);
```

4. Arguments just pass the name of the array (i.e., without square brackets) which is the reference to the first array element.

```
const int SIZE = 10;
int numbers[SIZE],
    used;
readNumbers(numbers, used);
```

5. Usually, the size of the array is a parameter to a function that takes the array as a parameter.

```
void readScores(int[] scores, int size, int & used); // note: scores is passed-by-reference,
// used is the num of elements used in the
array
```

6. Arrays cannot be returned by a function.

```
float[] readScores(); // Invalid
```

1. Write a function that reads known number of integers from user and store in an array.

```
void readNumbers(int numbers[], int & used)
{
    cout << "Please enter " << used << "numbers" << endl;
    for (int i = 0 ; i < used; i++)
    {
        cin >> numbers[i];
    }
}
```

2. Write a function that display the array elements (floats) on the screen

```
.....
.....
.....
.....
.....
.....
.....
.....
.....
```

3. Write a function that displays array elements (characters) in the reverse order of their entry.

```
.....
.....
.....
.....
.....
.....
.....
.....
.....
```


6. Write a function to find the highest and the lowest in an array; return both results to the calling function.

7. Write a function to determine whether two arrays are equal; return true if they are equal and false, otherwise.

8. Write a function that copy the array tests, to array scores.

```
const int SIZE = 10;
int tests[SIZE] = {0},
    scores[SIZE];

scores = tests;           //compilation error!
```

Note: the starting memory address of an array cannot be changed.

CH 7: Arrays (Part 3)

	A name for the function	Return type	parameters		Function call
A function that fills an array with a known number of elements					
A function that reads the number of elements want to be used and fill the array with that many elements.					
A function that doubles each array element					
A function that displays the difference					
A function that finds the availability of a given value in an array					
A function that finds the number of elements greater than a given number					
A function that sorts an array					

Common mistakes and right way to do them

	Invalid	Valid	comment
Array size allocator	int scores[];	int scores [5] = {79, 82, 91, 77, 84}; or int scores [5] = {79, 82, 91,}; or int scores [] = {79, 82, 91, 77, 84};	Must use either array size declarator or initialization list at array definition
Initialization	int scores[10] ; scores = {79, 82, 91, 77};	int scores[10] = {79, 82, 91, 77};	Values should be assigned along with the definition, otherwise, individual element should be accessed and assigned. scores[0] = 79; scores[1] = 82; scores[2] = 91; scores[3] = 77; for (int i =0; i < SIZE; i++) { cin >> scores[i]; }

Display an array	int scores[10] = {79, 82, 91, 77}; cout << scores;	for (int i = 0; i < 4; i++) { cout << scores[i] << " "; }	must access individual element to display an array.
Copy an array	int scores[10] = {79, 82, 91, 77}; int tests[10]; tests = scores;	for (int i = 0; i < 4; i++) { tests[i] = scores[i]; }	must access individual element to copy an array.
Return an array to the calling function.	int[] readScores();	void readScores(int [] scores, int size, int &used);	Must use array in parameter list to return the values back to the calling function. Usually array size (pass by value) and number of used elements (pass-by- depends) are passed as parameters. Note: in this example <i>size</i> is pass by value (read only), and <i>used</i> is pass-by-reference (assuming the function will read it from user).
arguments	const int SIZE = 10; int scores[SIZE], used; readScores(scores[10], SIZE, used);	const int SIZE = 10; int scores[SIZE], used; readScores(scores, SIZE, used);	

Write a program that fills an array with user defined number of elements and user defined elements (the scores of COSC 1550). Then, output the array to make sure the correct numbers are stored. Then add 5 to each array element and output the resulting array to make sure the numbers are correct. Count the number of elements that are lower than 50.

Welcome to my program!
Enter the number of students in your class (< 30)5

Please enter a score for each student in your class

1. 12
2. 34
3. 23
4. 67
5. 87

Here are the scores you entered
12 34 23 67 87

Here are the new scores
17 39 28 72 92

2 students of 5 have passed the course

Write a program that fills an array with user defined number of elements and user defined elements (the scores of COSC 1550). Then, display the array to make sure the correct numbers are stored. Then find each student has passed the course or not. Store these results in a separate array. Display this array on the screen too.

Multidimensional Arrays

Consider the following table

Freshman Enrollment (note that the following numbers are fake numbers)

		0	1	2	3	4
		Fall 2017	Fall 2016	Fall 2015	Fall 2014	Fall 2013
0	Accounting	17	13	19	43	39
1	Business	11	26	24	63	66
2	Computer Science	90	67	51	48	59
3	Economics	73	70	65	72	16
4	Entrepreneurship	55	67	47	34	17
5	Finance	68	21	43	51	25
6	Management	21	10	77	46	60
7	Marketing	13	43	30	60	10
8	Mathematics	31	35	45	60	45
9	Mobile Computing	44	11	13	32	70

A multidimensional array can be used to maintain the above set of data (note that all of the data are in the same type).

Multi-dimensional array definition

```
int enrollments[10][5];  
  
Or  
  
const int ROWS = 10,  
        COLUMNS = 5;  
  
int enrollments[ROWS][COLUMNS];
```

number of elements = $10 \times 5 = 50$

Accessing multi-dimensional array elements:

The individual element of an array is associated with two subscripts: row and column

enrollments	0	1	2	3	4
0	17	13	19	43	39
1	11	26	24	63	66
2	90	67	51	48	59
3	73	70	65	72	16
4	55	67	47	34	17
5	68	21	43	51	25
6	21	10	77	46	60
7	13	43	30	60	10
8	31	35	45	60	45
9	44	11	13	32	70

enrollments [0][0]

enrollments [3][0]

Array elements can be used as regular variables:

```
cout << enrollments[0][0] << endl;
```

output:

```
cout << enrollments[0][2] << endl;
```

output:

```
cout << enrollments[3][0] << endl;
```

output:

```
cout << enrollments[9][4] << endl;
```

output:

Using Loops (nested) to Step Through a multi-dimensional array

```
for (int row = 0; row < ROWS; row++)  
{  
    for (int col = 0; col < COLUMNS; col++)  
    {  
        cout << elements[row][col] << " ";  
    }  
}
```

Array Initialization

int hours [3][2] = {{1, 2}, {3, 4}, {5, 6}}

number of elements = 3 x 2 = 6

1	2
3	4
5	6

int hours [3][2] = {1, 2, 3, 4, 5, 6}

1	2
3	4
5	6

int hours [3][2] = {{1}, {3, 4}, {5}}

1	0
3	4
5	0

int hours [3][2] = {1, 3, 4, 5}

1	3
4	5
0	0

Functions with multi-dimensional arrays

1. The parameters show that the parameter is a multi-dimensional array by putting a two pairs of square brackets. The array parameter MUST contain the size declarator for the columns.

```
void displayEnrollement (int numbers [][][5], int rowUsed, int colUsed);
```

2. Arguments just pass the name of the array (i.e., without square brackets) which is the reference to the first array element.

```
const int ROW = 10,  
        COL = 5;  
  
int enrollment[ROW][COL],  
    rUsed,  
    cUsed;  
readNumbers(enrollment, rUsed, cUsed);
```