CSCI 1300 CS1: Starting Computing
Ashraf, Fleming, Correll, Cox, Fall 2019
Homework 4
Due: Saturday, October 5th, by 6 pm
(5 % bonus on the total score if submitted by 11:59 pm Oct. 4rd)

2 components (Moodle CodeRunner attempts, and zip file) must be completed and submitted by Saturday, October 5th, 6:00 pm for your homework to receive points.

---

0. Table of Contents

---

1. Objectives

- Understand and work with array operations: initialization, search
- Improve proficiency with loops and strings
- Passing an array to a function (“pass by reference” vs “pass by value”)
- Writing and testing C++ functions
- Test it in the Cloud9 IDE and submit it for grading on Moodle

---

2. Background

2.1 Arrays

An array is a *data structure* which can store primitive data types like floats, ints, strings, chars, and booleans.

Arrays have both a **type** and a **size**.

- **How to Declare Arrays**

```
data_type array_name[declared_size];
bool myBooleans[10];
string myStrings[15];
int myInts[7];
```

- **How to Initialize Arrays (Method 1)**

```
bool myBooleans[4] = {true, false, true, true};
```

If you do not declare the size inside the square brackets, the array size will be set to however many entries you provide on the right.

```
bool myBooleans[] = {true, false, true}; // size = 3
```

Note: the size specified in the brackets needs to match the number of elements you wrote in the curly brackets.

Example 1: When the specified size is larger than the actual number of elements, the elements provided in the curly brackets will be the first several elements in the array, while the additional elements will be filled with default values. If it's an integer/double array, the default values are zero, while if it's a string array, the default values are empty strings.

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

int main() {
    int intArray[5] = {1,2,3};
    for (int i = 0; i < 5; i ++) {
        cout << intArray[i] << " ";
    }
}
```

Output:

```
1 2 3 0 0
```

Example 2: When the specified size is smaller than the actual number of elements, there will be a compilation error.

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

int main() {
    int intArray[3] =
            {1,2,3,4,5};
}
```

Output (Error message):

```
error: excess elements in array
initializer
int intArray[3] = {1,2,3,4,5};
                          ^

1 error generated.
```

- **How to Initialize Arrays (Method 2)**

  You can also initialize elements one by one using a for or a while loop:

```
int myInts[10];
int i = 0;
while (i < 10) {
    myInts[i] = i;
    i++;
}
//{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
```

- **How to Access Elements in an Array**

  We have essentially already had practice with accessing elements in an array, as in C++, strings are array of characters.

You can access elements in arrays using the same syntax you used for strings:

```
string greetings[] = {"hello", "hi", "hey", "what's up?"};
cout << greetings[3] << endl;
```

"hello"	"hi"	"hey"	**"What's up?"**
0	1	2	**3**

Arrays can be iterated in the same way we iterated over strings last week. Below we are iterating through an array of strings:

```
string greetings[] = {"hello", "hi",
"hey", "what's up?"};
int size = 4;
int i = 0;
while (i < size){
      cout << greetings[i] << endl;
      i++;
}
```

Output:

```
hello
hi
hey
what's up?
```

2.2 Passing arrays to functions

- **Passing By Value**

  Up until now, when calling functions, we have always *passed by value.* When a parameter is passed in a function call, a new variable is declared and initialized to the value *passed* in the function call.

  Observe that the variable x in **main** and variable x in **addOne** are *separate* variables in memory. When **addOne** is called with x on line 9, it is the *value* of x (i.e. 5) that is *passed* to the function. This *value* is used to initialize a new variable x that exists only in **addOne**'s scope. Thus the value of the variable x in main's scope remains unchanged even after the function **addOne** has been called.

```
1      void addOne(int x){
2          x = x + 1;
3          cout << x << endl;
```

Output:

```
4       }
5
6       int main(){
7          int x = 5;
8          cout << x << endl;
9          addOne(x);
10         cout << x << endl;
11      }
```

```
5
6
5
```

- **Passing By Reference**

  Arrays, on the other hand, are *passed by reference* (to the original array's location in the computer's memory). So, when an array is passed as a parameter, the original array is used by the function.

  Observe that there is only *one* array X in memory for the following example. When the function **addOne** is called on line 9, a *reference* to the original array X is *passed* to **addOne**. Because the array X is *passed by reference*, any modifications done to X in **addOne** are done to the original array. These modifications persist and are visible even after the flow of control has exited the function and we return to main.

```
1       void addOne(int X[]){
2           X[0] = X[0] + 1;
3           cout << X[0] << endl;
4       }
5       int main(){
6          int X[4] = {1, 5, 3, 2};
7          cout << X[0] << endl;
8          addOne(X);
9          cout << X[0] << endl;
10      }
```

Output:

```
1
2
2
```

When we pass a one-dimensional array as an argument to a function we also provide its length. For two-dimensional arrays, in addition to providing the length (or number of rows), we will also assume that we know the length of each of the subarrays (or the number of columns). A function taking a two-dimensional array with 10 columns as an argument then might look something like this:

```
void twoDimensionalFunction(int matrix[][10], int rows){ … }
```

---

3. Submission Requirements

All three steps must be fully completed by the submission deadline for your homework to be graded.

1. ***Work on questions on your Cloud 9 workspace:*** You need to write your code on Cloud 9 workspace to solve questions and test your code on your Cloud 9 workspace before submitting it to Moodle. (Create a directory called **hmwk4** and place all your file(s) for this assignment in this directory to keep your workspace organized)

2. ***Submit to the Moodle CodeRunner:*** Head over to Moodle to the link **Homework 4 CodeRunner**. You will find one programming quiz question for each problem in the assignment. Submit your solution for the first problem and press the Check button. You will see a report on how your solution passed the tests, and the resulting score for the first problem. You can modify your code and re-submit (press *Check* again) as many times as you need to, up until the assignment due date. Continue with the rest of the problems.

3. ***Submit a .zip file to Moodle:*** After you have completed all 6 questions from the Moodle assignment, zip all 6 solution files you compiled in Cloud9 (one cpp file for each problem), and submit the zip file through the **Homework 4** link on Moodle.

---

4. Rubric

Aside from the points received from the **Homework 4 CodeRunner** quiz problems, your TA will look at your solution files (zipped together) as submitted through the **Homework 4** link on Moodle and assign points for the following:

Style, Comments, Algorithm (10 points):

Style:

- Your code should be well-styled, and we expect your code to follow some basic guidelines on whitespace, naming variables and indentation, to receive full credit. Please refer to the **CSCI 1300 Style Guide** on Moodle.

Comments:

- Your code should be well-commented. Use comments to explain what you are doing, especially if you have a complex section of code. These comments are intended to help other developers understand how your code works. These comments should begin with two backslashes (`//`) or the multi-line comments (`/* … comments here… */`) .
- Please also include a comment at the top of your solution with the following format:

```
// CS1300 Fall 2019
// Author: my name
// Recitation: 123 – Favorite TA
// Homework 4 - Problem # ...
```

Algorithm:

- Before each function that you define, you should include a comment that describes the inputs and outputs of your function and what algorithms you are using inside the function.
- This is an example C++ solution. Look at the code and the algorithm description for an example of what is expected.

Example 1:

```
/*
 * Algorithm: convert money from U.S. Dollars (USD) to Euros.
 *    1. Take the value of number of dollars involved
 *       in the transaction.
 *    2. Current value of 1 USD is equal to 0.86 euros
 *    3. Multiply the number of dollars got with the
 *       currency exchange rate to get Euros value
 *    4. Return the computed Euro value
 * Input parameters: Amount in USD (double)
 * Output (prints to screen): nothing
 * Returns: Amount in Euros (double)
 */
```

Example 2:

```
double convertUSDtoEuros(double dollars)
{
     double exchange_rate = 0.86; //declaration of exchange
rate
     double euros = dollars * exchange_rate; //conversion
     return euros; //return the value in euros
}
```

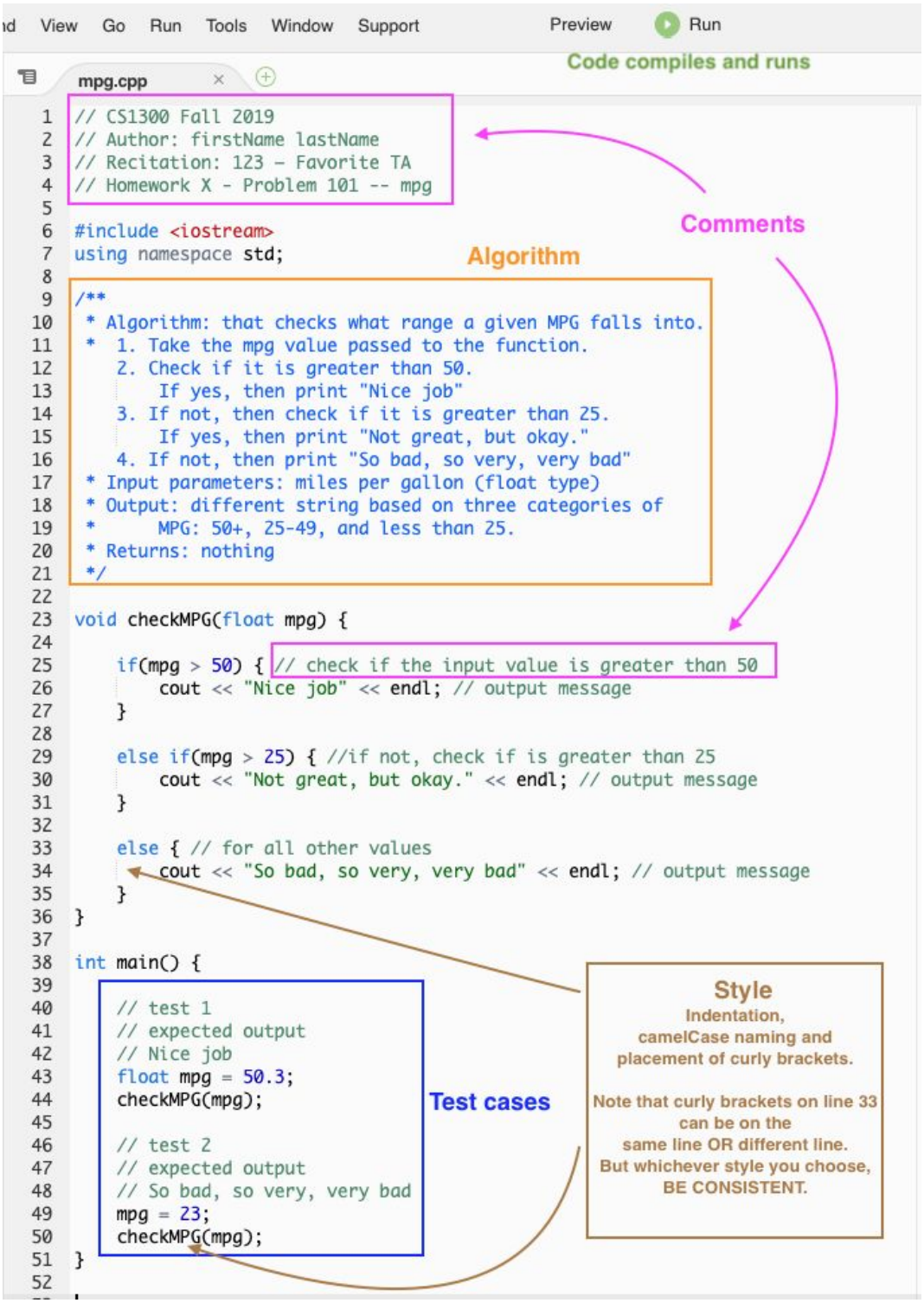
The algorithm described below does not mention in detail what the algorithm does and does not mention what value the function returns. Also, the solution is not commented. This would work properly, but would not receive full credit due to the lack of documentation.

```
/*
 * conversion
 */
double convertUSDtoEuros(double dollars)
{
      double euros = dollars * 0.86;
      return euros;
}
```

Test Cases (20 points)

1. *Code compiles and runs* (**6 points**):
    - The zip file you submit to Moodle should contain **6** full programs (with a `main()` function), saved as .cpp files. It is important that your programs can be compiled and run on Cloud9 with no errors. The functions included in these programs should match those submitted to the CodeRunner on Moodle.

2. *Test cases* (**14 points**):
   For this week's homework, all 6 problems are asking you to create a function. In your solution file for each function, you should have at least 2 test cases present in their respective `main()` function, for a total of 12 test cases (see the diagram on the next page). Your test cases should follow the guidelines, **Writing Test Cases**, posted on Moodle under Week 3.




```
1  // CS1300 Fall 2019
2  // Author: firstName lastName
3  // Recitation: 123 - Favorite TA
4  // Homework X - Problem 101 -- mpg
5
6  #include <iostream>
7  using namespace std;
8
9  /**
10  * Algorithm: that checks what range a given MPG falls into.
11  *   1. Take the mpg value passed to the function.
12     2. Check if it is greater than 50.
13         If yes, then print "Nice job"
14     3. If not, then check if it is greater than 25.
15         If yes, then print "Not great, but okay."
16     4. If not, then print "So bad, so very, very bad"
17  * Input parameters: miles per gallon (float type)
18  * Output: different string based on three categories of
19  *      MPG: 50+, 25-49, and less than 25.
20  * Returns: nothing
21  */
22
23 void checkMPG(float mpg) {
24
25     if(mpg > 50) { // check if the input value is greater than 50
26         cout << "Nice job" << endl; // output message
27     }
28
29     else if(mpg > 25) { //if not, check if is greater than 25
30         cout << "Not great, but okay." << endl; // output message
31     }
32
33     else { // for all other values
34         cout << "So bad, so very, very bad" << endl; // output message
35     }
36 }
37
38 int main() {
39
40     // test 1
41     // expected output
42     // Nice job
43     float mpg = 50.3;
44     checkMPG(mpg);
45
46     // test 2
47     // expected output
48     // So bad, so very, very bad
49     mpg = 23;
50     checkMPG(mpg);
51 }
52
```

5. Problem Set

Note: To stay on track for the week, we recommend to finish/make considerable progress on problems 1-3 by Wednesday. Students with recitations on Thursday are encouraged to come to recitation with questions and have made a start on all of the problems.

Problem 1 (10 points): arrayPilgrimage

Write a program `arrayPilgrimage.cpp` to **declare and populate** the four arrays listed below. For testing in Coderunner, you will paste the `main()` function from your program.

- `temps` - an array of 10 floating point numbers (type `double`) initialized with -459.67 (absolute zero in Fahrenheit)
- `colors` - an array of the strings “Red”, “Blue”, “Green”, “Cyan”, and “Magenta”, in that order.
- `sequence` - an array of the first 100 positive integers in order; 1, 2, 3, 4, … etc.
- `letters` - an array of all uppercase and lowercase characters in order, A, a, B, b, C, c, … **Hint:** the ASCII table will be helpful here!

In order to test that you correctly created and populated the arrays, print the values of each of their elements, in order, one element per line. **Hint:** use a loop to traverse each array.

Expected output:

```
-459.67
-459.67
…
-459.67

Red
Blue
…
Magenta

1
2
…
100

A a
B b
…
Z z
```

In Cloud9 the file should be called **arrayPilgrimage.cpp** and it will be one of the files you need to zip together for the **Homework 4** on Moodle.

Don't forget to head over to Moodle to the link **Homework 4 CodeRunner**. For Problem 1, in the Answer Box, paste the entire program. Press the Check button. You can modify your code and re-submit (press Check again) as many times as you need to.

Problem 2 (10 points): printArrReverse

Write a function named **printArrReverse** that has an integer array and length of the array as parameters and prints the numbers in the given array in the reverse order.

- Your function **MUST** be named **printArrReverse**
- Your function has two parameters in the following order:
  - An array of type `int`
  - The number of elements in the array, type `int`
- Your function should NOT return anything
- Your function must print each number in a new line

Examples:

Function calls	Output

`int arr[] = {0,1,2,3,4,5,6,7,8,9};`
`int len = 10;`
`printArrReverse(arr, len);`
	9
	8
	7
	6
	5
	4
	3
	2
	1
	0

`int arr[] = {1, 9, 7};`
`int len = 3;`
`printArrReverse(arr, len);`
	7
	9
	1

In Cloud9 the file should be called **printArrReverse.cpp** and it will be one of the files you need to zip together for the **Homework 4** on Moodle.

Don't forget to head over to Moodle to the link **Homework 4 CodeRunner**. For Problem 2, in the Answer Box, paste only your function definition, not the entire program. Press the Check button. You can modify your code and re-submit (press Check again) as many times as you need to.

Problem 3 (10 points): stats

Write a function named **stats**, that takes an array and the number of elements in the array as arguments. It must compute and print the minimum value in the array, the maximum value in the array and the average value of all the values in the array.

- Your function **MUST** be named **stats**
- Your function has two parameters in the following order:
  - An array of type `double`
  - The number of elements in the array, type `int`
- Your function should NOT return anything.
- You function should output the values with 2 decimal places of precision (Hint: use `fixed` and `setprecision()`

Examples:

Function calls	Output

`double arr[] = {0,1,2,3,4,5,6,7,8,9};`
`int len = 10;`
`stats(arr, len);`
	`Min: 0.00`
	`Max: 9.00`
	`Avg: 4.50`

`double arr[] = {1.4, 9.8, 2.6};`
`int len = 3;`
`stats(arr, len);`
	`Min: 1.40`
	`Max: 9.80`
	`Avg: 4.60`

In Cloud9 the file should be called **stats.cpp** and it will be one of the files you need to zip together for the **Homework 4** on Moodle.

Don't forget to head over to Moodle to the link **Homework 4 CodeRunner**. For Problem 3, in the Answer Box, paste only your function definition, not the entire program. Press the Check button. You can modify your code and re-submit (press Check again) as many times as you need to.

Problem 4 (5 points): swapFirstLast

Write a function **swapFirstLast** that swaps the first and last elements of the given array of integers.

- Your function MUST be named **swapFirstLast**
- Your function takes two parameters in the following order:
  - An array of integers

    - The number of elements in the array, type `int`
- Your function should swap the first and last elements of the array
- Your function should NOT return anything
- Your function should NOT print anything

Examples:

Function calls	Output

`int arr1[3] = {10, 20, 30};`
`swapFirstLast(arr1, 3);`
`printArray(arr1, 3);`
	30
	20
	10

`int arr1[5] = {5, 4, 3, 2, 1};`
`swapFirstLast(arr1, 5);`
`printArray(arr1, 5);`
	1
	4
	3
	2
	5

`int arr2[1] = {10};`
`swapFirstLast(arr2, 1);`
`printArray(arr2, 1);`
	10

`int arr2[] = {};`
`swapFirstLast(arr2, 0);`
`printArray(arr2, 0);`	

In Cloud9 the file should be called **swapFirstLast.cpp** and it will be one of the files you need to zip together for the **Homework 4** on Moodle.

Don't forget to head over to Moodle to the link **Homework 4 CodeRunner**. For Problem 4, in the Answer Box, paste only your function definition, not the entire program. Press the Check button. You can modify your code and re-submit (press Check again) as many times as you need to.

Problem 5 (10 points): getWordCount

A space (`' '`) is a single *character* in C++, just like `'a'`, `'A'` , `'5'`, and `'%'` (for example). In English, a space is used to separate individual words. Write a function that takes an English sentence and returns the number of words in the sentence.

- Your function should take **one** parameter:

    - a `string` parameter for the sentence
- Your function **should RETURN** the number of words in the sentence.
- Your function should not print anything.
- Your function *MUST* be named **getWordCount**.
- ***Note****: You can assume there is exactly one single space between any two words.*

Examples:

- Input parameter: `""` (an empty string) ➔ return `0`;
- Input parameter: `"Go"` ➔ return `1`;
- Input parameter: `"I went"` ➔ return `2`;
- Input parameter: `"Colorless green ideas dream furiously"` ➔ return `5`;
- Input parameter: `"The rat the cat the dog bit chased escaped"` ➔ return `9`;

In Cloud9 the file should be called **getWordCount.cpp** and it will be one of the files you need to zip together for the **Homework 4** on Moodle.

Don't forget to head over to Moodle to the link **Homework 4 CodeRunner**. For Problem 6, in the Answer Box, paste **only your function definition, not the entire program**. Press the Check button. You can modify your code and re-submit (press Check again) as many times as you need to.

Problem 6(25 points): split

Write a function **split** which takes four input arguments: a string to be split, a character to split on ("a delimiter"), an array of strings to fill with the split pieces of the input string, and an integer representing the maximum number of split string pieces. The function will split the input string in to pieces separated by the delimiter, and populate the array of strings with the split pieces up to the provided maximum number of pieces. Your function will return the number of pieces the string was split into.

- Your function should be named **split**
- Your function takes four parameters in the following order:
    - The **string** to be split.
    - A delimiter **character**, which marks where the above string should be split up.
    - An **array of string**, which you will use to store the split-apart string pieces.
    - The **int** length of the given array
- Your function should **return** the number of pieces the string was split into as an integer.
- Your function does not print anything.
- If the string is split into more pieces than the array of string can hold (more than the indicated length), your function should fill only as many words as it can, and return -1.

Example: in the following examples, `printArray` function will output all the elements in the array, one per line.

Function calls	Output

`string words[6];`
`cout << split("one small step",' ',words,6) << endl;`
`printArray(words);`
	`3`
	`one`
	`small`
	`step`

`string words[6];`
`cout << split(" one small step  ",' ',words,6) << endl;`
`printArray(words);`
	`3`
	`one`
	`small`
	`step`

`string words[6];`
`cout << split("cow/big pig/fish",'/',words,6) << endl;`
`printArray(words);`
	`3`
	`cow`
	`big pig`
	`fish`

`string words[6];`
`cout << split("cow/big pig//fish",'/',words,6) << endl;`
`printArray(words);`
	`3`
	`cow`
	`big pig`
	`fish`

`string words[6];`
`cout << split("unintentionally",'n',words,6) << endl;`
`printArray(words);`
	`5`
	`u`
	`i`
	`te`
	`tio`
	`ally`

`string words[2];`
`cout << split("one small step",' ',words, 2) << endl;`
`printArray(words);`
	`-1`
	`one`
	`small`

In Cloud9 the file should be called **split.cpp** and it will be one of the files you need to zip together for the **Homework 4** on Moodle.

Don't forget to head over to Moodle to the link **Homework 4 CodeRunner**. For Problem 7, in the Answer Box, paste **only your function definition, not the entire program**. Press the Check button. You can modify your code and re-submit (press Check again) as many times as you need to.

Extra credit (15 points): shiftElements

Write a function **shiftElements** that takes an array of integers, number of elements n, and number of shifts k. The function should shift each element in the array by k positions. If k is greater than n, each element should be shifted by k mod n positions.

- Your function MUST be named **shiftElements**.
- Your function has three parameters in the following order:
  - An array of integers
  - The number of elements in the array, type `int`
  - The number of shits, an integer
- Your function should shift the elements in the array
- Your function should NOT return anything
- Your function should print “No shift”, if the number of shifts is negative or 0.

Examples:

Function calls	Output	Explanation

`int arr1[3] = {10, 20, 30};`
`shiftElements(arr1, 3, 1);`
`printArray(arr1, 3);`
	30
	10
	20
		Each element is shifted by 1 position. Note how the last element is moved to the first position.

`int arr1[3] = {10, 20, 30, 40, 50};`
`shiftElements(arr1, 5, 3);`
`printArray(arr1, 5);`
	30
	40
	50
	10
	20
		Each element is shifted by 3 positions.

`int arr1[3] = {10, 20, 30, 40, 50};`
`shiftElements(arr1, 5, 7);`
`printArray(arr1, 5);`
	40
	50
	10
	20
	30
		Number of shifts is greater than 7, so shift by 7 mod 5 = 2 positions.

`int arr1[3] = {10, 20, 30, 40, 50};`
`shiftElements(arr1, 5, -2);`
`printArray(arr1, 5);`
	No shift
	10
	20
	30
		Number of shifts is negative

```
40
50
```

Don't forget to head over to Moodle to the link **Homework 4 CodeRunner**. For Problem 5, in the Answer Box, **paste only your function definition**, not the entire program. Press the Check button. You can modify your code and re-submit (press Check again) as many times as you need to.

6. Homework 4 checklist

Here is a checklist for submitting the assignment:

1. Complete the code **Homework 4 CodeRunner**
2. Submit one zip file to **Homework 4**. The zip file should be named, **hmwk4_lastname.zip**. It should have the following 7 files:
    - arrayPilgrimage.cpp
    - printArrReverse.cpp
    - stats.cpp
    - swapFirstLast.cpp
    - getWordCount.cpp
    - split.cpp

7. Homework 4 points summary

Criteria	**Pts**
CodeRunner (problem 1 - 6)	70
Style, Comments, Algorithms	10
Test cases	20
Recitation attendance (week 6)*	-30
Total	100
Extra credit: shiftElements	15
5% early submission bonus	+5%

* if your attendance is not recorded, you will lose points. Make sure your attendance is recorded on Moodle.