Programming Assignment #3

Practice with pointers and dynamic memory allocation

CS 2308.003 and 004 Fall 2018
Instructor: Jill Seaman

Due: Monday, 10/08/2018: upload electronic copy by 9:00am.

Problem:

Write a C++ program that will implement and test the functions described below that use pointers and dynamic memory allocation.

The Functions:

You will write functions for the 4 problems described below. Then you will call them from the main function, to demonstrate their correctness.

1. **leftCircularShift**: takes an array of integers and its size as arguments. It should do a left circular shift of the array of integers. This function should change the order of the elements in the array argument by moving them one position to the left, and moving the first element to the last position. **Do not use square brackets anywhere in the function, not even the parameter list (use pointer notation instead).**

   Example: leftCircularShift([1 2 3 4 5]) -> [2 3 4 5 1]

2. **litersForKm**:
   `double litersForKm(double mpg, double km, double *lp100km);`
   Implement a function that would be helpful for travelers to Europe. Given how many miles your car could run on one gallon in the 'mpg' parameter, calculate how many liters you car needs to run 100 km (this measure is used instead of mpg in Europe) and assign it to the variable pointed to by the 'lp100km' parameter. Also, compute the amount of liters of fuel required to travel the kilometers given by 'km' parameter.

   Example: `litersForKm(25.0, 10.0, _) ->  0.94, lp100km: 9.41`

   The following constants are required:
   1 gallon is 3.785411784 liters
   1 mile is 1.609344 kilometers

   Here is a link to a calculator doing these operations https://www.calculateme.com/gas-mileage/us-mpg-to-liters-per-100-km

3. **subArray**: takes the address of two elements in an array. It creates a new array that is a **copy** of the elements from the original array starting at the first element and ending at the second element. For example, subArray(&aa[5],&aa[8]) would return a new array containing only the elements aa[5], aa[6], aa[7], and aa[8].

   You must define subArray as follows:

   Add the code for the duplicateArray function from the lecture slides for chapter 9 to your program. Add the code for the subArray function given below to your program. Fill in the blanks with expressions so that the function subArray behaves as described above (you may use additional variables in subArray as desired).

```
int *subArray(int *begin, int *end) {
    int *result = duplicateArray(__________, ___________);
    return result;
}
```

   DO NOT alter duplicateArray, DO NOT alter subArray as defined above.

4. **arrayOfPtrs**: takes an array of string and its size as arguments. It should create a new array of the same size whose elements are pointers to strings (addresses). It should copy the address of each element from the original array into the corresponding element of the new array. The function should return a pointer to the new array (hint: the return type will be `string**`, a pointer to a pointer to a string, because the elements of the new array are pointers to string, and the result is a pointer to the first element of this array).
   outputArray: takes an array of pointers to string and its size as arguments. It should iterate over the array and output the string pointed to by each element. Call this function from main to output the results of calling arrayOfPtrs.

Output:

Test these four(+1) functions using the main function as a driver. The driver should pass **constant** test data as arguments to the functions. Select appropriate test data for each function and then call that function using the test data. For each function, you should output the following: (1) a label indicating which function is being tested, (2) the test data, (3) the expected results, and (4) the actual results. For the test data and Expected result, you should hard code the output values (use string literals containing the numeric values), for the Actual result, use the actual values returned/altered by the function. No named constants are required, and the program should do no input.

```
Testing leftCircularShift:
test data array 1: 1 2 3 4 5 6 7 8
Expected result: 2 3 4 5 6 7 8 1
Actual result:   2 3 4 5 6 7 8 1
shift again:
```

```
Expected result: 3 4 5 6 7 8 1 2
Actual result:   3 4 5 6 7 8 1 2

Testing litersForKm:
testing for 25.0mpg and 10.0km
Expected result: 9.41 Lp100km and 0.94 liters
Actual results : 9.41 and 0.94 liters
testing for 33.60mpg and 700.0km
Expected result: 7.00 Lp100km and 49.00 liters
Actual results : 7.00 and 49.00 liters

testing subArray:
test data: 1 2 3 4 5 6 7 8 3
start: index 3 finish: index 5
Expected result: 4 5 6
Actual result:   4 5 6

testing arrayOfPtrs:
test data: echo charlie delta bravo alpha
Expected result: echo charlie delta bravo alpha
Actual result:   echo charlie delta bravo alpha
```

RULES:

- DO NOT change the names of the functions!
- DO NOT do any output from the functions (except outputArray)!
- DO NOT get any input from the user!! Use literal constants for test values!!
- Output the float/double values to 2 decimal places.

NOTES:

- This program must be done in a **Linux or Unix** environment, using a command line compiler like g++. Do not use codeblocks, eclipse, or Xcode to compile.
- Your program **must compile** and run, otherwise you will receive a score of 0.
- It is your responsibility to fully test your functions. They must work for ANY valid input. The main function must have at least one test case for each function.
- For litersForKm, compute the value of the call to litersForKm BEFORE you output it:

```
double z = litersForKm(…….);
    cout << z << .....;
```

- You do not need to use **named** constants for your test data (or array sizes) in this assignment, but you DO need to follow the rest of the style guidelines including function definition comments.

- Your program should release any dynamically allocated memory when it is finished using it.

- I recommend using a function that displays the values of an int array on one line, separated by spaces, for displaying test arrays and results.

Logistics:

Name your file **assign3_xxxxx.cpp** where xxxxx is your TX State NetID (your txstate.edu email id).

There are two steps to the turn-in process:

1. Submit an **electronic copy** using the Assignments tool on the TRACS website for this class (tracs.txstate.edu).

2. Submit a **printout** of the source file at the beginning of class, the day after the assignment is due. Please **print your name on top of the front page**, and staple if there is more than one page.

See the assignment policy on the course website (cs.txstate.edu/~js236/cs2308) for more details, including deadlines, penalties, and where to submit printouts outside of class.