Programming Assignment #3

Practice with pointers and dynamic memory allocation

CS 2308.255 + CS5301 Spring 2020
Instructor: Jill Seaman

Due: Monday, 3/2/2020: upload electronic copy by 11:59pm!

Problem:

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

The Functions:

You will write the five functions 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. **sort2withSum**: The following function uses reference parameters. Rewrite the function so it uses pointers instead of reference variables. When you test this function from the main program, demonstrate that it changes the values of the variables passed into it.

```
int sort2withSum (int &x, int &y) {
    if (x > y) {
        int temp = x;
        x = y;
        y = temp;
    }
    return x + y;
}
```

3. **resize**: takes an int array and the array's size as arguments. It should create a new array that is twice the size of the argument array. The function should copy

the contents of the argument array to the new array, and initialize the unused elements of the new array with -1. The function should return a pointer to the new array.

4. **concatenate**: takes two int arrays and the arrays' sizes as arguments (that's 4 arguments). It should create a new array big enough to store both arrays. Then it should copy the contents of the first array to the new array, and then copy the contents of the second array to the new array in the remaining elements, and return a pointer to the new array.

5. **subArray**: takes an int array, a start index and a length as arguments. It creates a new array that is a copy of the elements from the original array starting at the start index, and has length equal to the length argument. For example, subArray(aa,5,4) 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 Unit 3 (slide 24) 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.

```
int *subArray (int *array, int start, int length) {
    int *result = duplicateArray(__________, ___________);
    return result;
}
```

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

Output:

Test these five 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 four lines: a label indicating which function is being tested, the test data, the expected results, and 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 results, use the actual values returned/altered by the function.

```
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 sort2withSum
test data: a:8 b:5
Expected result: 13  a: 5  b: 8
Actual results : 13  a: 5  b: 8

testing resize:
test data: 1 2 3 4 5 6 7 8 9 0
Expected result: 1 2 3 4 5 6 7 8 9 0 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
Actual result:   1 2 3 4 5 6 7 8 9 0 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1

testing concat:
test data: 1 2 3 4 5 6 7 8 9 0
 and 11 22 33 44 55
Expected result: 1 2 3 4 5 6 7 8 9 0 11 22 33 44 55
Actual result:   1 2 3 4 5 6 7 8 9 0 11 22 33 44 55

testing subArray:
test data: 1 2 3 4 5 6 7 8 9 0
start: 5 length: 4
Expected result: 6 7 8 9
Actual result:   6 7 8 9
```

RULES:

- DO NOT change the names of the functions!
- DO NOT do any output from the functions (only from main)!
- DO NOT do any input at all!

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.

- 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 `sort2withSum`, compute the value of the function call BEFORE you output it:

```
int z = sort2withSum(.......);
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).

Submit an **electronic copy** using the Assignments tool on the Canvas website for this class (canvas.txstate.edu).

See the assignment policy on the course website (cs.txstate.edu/~js236/cs2308) for more details, including late deadlines and penalties.