Programming Assignment #3

Practice with pointers and dynamic memory allocation

CS 2308.255 + CS5301 Spring 2019
Instructor: Jill Seaman

Due: Monday, 3/4/2019: upload electronic copy by 11:55pm.

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. **minimum**: takes an int array and the array's size as arguments. It should return the minimum value of the array elements. **Do not use square brackets anywhere in the function, not even the parameter list (use pointers instead).**

2. **swapTimesTen**: 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 swapTimesTen (int &x, int &y)
{
   int temp = x;
   x = y * 10;
   y = temp * 10;
   return x + y;
}
```

3. **doubleArray**: 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 first half of the new array, and the contents of the argument array each multiplied by 2 to the second half of the new array. The function should return a pointer to the new array.

4. **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]. The function should return a pointer to the new array. Do not call any other functions from this function. Hint: modify the code from duplicateArray.

5. **subArray2**: takes an int array, a start index and a length as arguments. It behaves exactly like subArray, however, you must define subArray2 as follows:

   Add the code for the definition of the duplicateArray function from the lecture slides for Unit 3 (or from the textbook) to your program. Add the code for the subArray2 function given below to your program. Fill in the blanks with expressions so that the function subArray2 behaves as the first subArray.

```
int *subArray2 (int *array, int start, int length) {
    int *result = duplicateArray(__________, ___________);
    return result;
}
```

   DO NOT alter duplicateArray, DO NOT alter subArray2 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 minimum:
test data: 1 2 3 4 5 6 7 -8 9 0
Expected minimum: -8
Actual minimum:   -8
test data: 1 2 3 4 5 16 7 8 9 0
Expected minimum: 0
Actual minimum:   0

testing swapTimesTen
Expected result: 80  a: 50  b: 30
Actual results : 80  a: 50  b: 30
```

```
testing doubleArray:
test data: 1 2 3 4 5 6 7 8 9
Expected result: 1 2 3 4 5 6 7 8 9 2 4 6 8 10 12 14 16 18
Actual result:   1 2 3 4 5 6 7 8 9 2 4 6 8 10 12 14 16 18

testing subArray:
test data: 1 2 3 4 5 6 7 8 9 10
start: 5 length: 4
Expected result: 6 7 8 9
Actual result:   6 7 8 9

testing subArray2:
test data: 1 2 3 4 5 6 7 8 9 10
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 from the user!! Use constants for test values!!

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 swapTimesTen, compute the result of the function call BEFORE you output it:

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

2. Submit a printout of the source file at the beginning of class, the day 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 turn-in policy on the course website (cs.txstate.edu/~js236/cs2308) for more details, including deadlines, penalties, and where to submit printouts outside of class.