Programming Assignment #3

Practice with pointers and dynamic memory allocation

CS 2308.001 and 003, Fall 2013
Instructor: Jill Seaman

Due:
section 001: in class **Wednesday, 10/2/2013** (upload electronic copy by 3:00pm)
section 003: in class **Thursday, 10/3/2013** (upload electronic copy by 10:30pm)

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. **maximum**: takes an int array and the array's size as arguments. It should return the maximum value of the array elements. Do not use square brackets ANYWHERE in the function (use pointers instead). Extra challenge: Do not use the loop variable in the body of the loop.

2. **oddSwap**: 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 oddSwap (int &x, int &y) {
   int temp = x;
   x = y * 5;
   y = temp * 5;
   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 (Note: you could use a modified version of this function to make your inventory array in PA#2 bigger when it gets to 100 elements).

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**: Define subArray as follows:

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

Add the code for duplicateArray from the lecture slides for chapter 9 (slide 24). Fill in the blanks with expressions so that the function subArray behaves as follows:

It 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].

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 may 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 maximum:
test data: 1 2 3 4 5 6 7 -8 9 0
Expected maximum: 9
Actual maximum:   9

testing oddSwap
test data: a:3 b:5
Expected result: 40  a: 25  b: 15
Actual results : 40  a: 25  b: 15

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 should be developed using a Linux or Unix environment.
- 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 you submit must have at least one test case for each function.
- For oddswap, compute the value of the call to oddswap BEFORE you output it:

```
int z = oddswap(.......);
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.
- Do you not use any features of C++ that we have not yet covered in class (use features from Chapters 1-9 only.

Logistics:

Name your file **assign3_xxxxx.cpp** where xxxxx is your TX State NetID (your txstate.edu email id). The file name should look something like this: assign3_js236.cpp

There are two steps to the turn-in process:

1. Submit an electronic copy using the Assignments tool in TRACS no later than one hour before class the day the assignment is due (see top of page 1).

2. Submit a printout of the file at the beginning of class, the day the assignment is due. Make sure your name is on it.

   If you are unable to turn a printout in during class, you have until 5pm on the day the assignment is due to turn it in to the computer science department office (Nueces 247). They will stamp it and put it in my mailbox. DO NOT slide it under my office door.