Assg 12: Positive Ints Library

CSci 515 Spring 2015

2015-04-21

Dates:

Due: Tuesday April 28, by Midnight

Objectives

- Understand the relationship between arrays and pointers in C
- Use pointers to an array of items, and process them.
- Get more practice with more advanced sentinel based processing.

Description

In this assignment you will extend the basic functionality from your lab 12 work. In this assignment, you will be implementing further functions that can be used to process pointers to arrays of positive integers that are terminated with a `-1` sentinel character.

Perform the following tasks:

1. Expand the functionality of your `posintcmp()` function to be able to handle arrays of positive integers of differing lengths. For the `strlen()` function, if the string are equal up to position `n`, but one string is longer than the other, the longer string should be considered as occurring after shorter one. For example if `int* p1Ptr = {4, 3, -1}` and `int* p2Ptr = {4, 3, 2, -1}` then `posIntCmp(p1Ptr, p2Ptr)` should return a `-1`, and `posIntCmp(p2Ptr, p1Ptr)` should return a `1`.

2. Implement a `posintcpy()` function. This function takes a destination as its first parameter and a source as its second parameter (both of type pointer to an array of positive integers). This function is a void function, it does not return any explicit result. This function assumes that there is enough space allocated for the destination so that all of the values from the source list can be copied successfully. This function should cause all of the values from the source array of positive integers to be copied to the destination array space (including the `-1` terminating sentinel).

3. Implement a `posintfind()`. This function is similar to the `strchr` function. This function takes a pointer to an array of positive integers as its first parameter, and a single integer as its second parameter (the value to be searched for). This function should perform a linear search of the array of positive integers, and return the index of the first location where the value being searched for is found. This function will return a `-1` if the value being searched for is not found in the array of positive integers.

4. Create a function called `posintcat()`. This function is similar to the `strcat()` function. This function takes two pointers to arrays of positive integers null terminated with a `-1` sentinel character. This is a void function. The first parameters works as a source, and the second as the destination. However, unlike the copy function, this function should concatenate the source array values on to the end of the destination array. You can assume that the destination array has enough extra allocated space in order to hold the values that will be appended to the end. For example, if `int dest[5] = {1, 2, -1}` and `int src[] = {3, 4, -1}`, then the result of calling `posintcat(src, dest)` will be that `dest` now has the values `{1, 2, 3, 4, -1}`. **Extra Credit:** much of the functionality of this function has already been accomplished in other functions. Reuse functions like `posintlen` and `posintcpy` to implement this function.

5. Demonstrate all of your functions in your `main()` function. Make sure you adequately test and demonstrate the correct working of all of your functions.

Here is an example output from running a correct implementation of this assignment:

```
List 1: { 4, 2, 7 }
List 2: { 4, 2, 7, 3 }
Result of posintcmp(list1, list2): -1
Result of posintcmp(list2, list1): 1

List 3 (an empty list with enoough room to hold 10 values): {  }
After copy of list2 to list3, list3 = { 4, 2, 7, 3 }

Result of searching for 4 in List 2: 0
Result of searching for 7 in List 2: 2
Result of searching for 8 in List 2: -1

List 4 has enough room for 10 values, but
   currently contains only 4: { 9, 10, 15, 22 }
Result of concatenating List 2 onto end of
   List 4: { 9, 10, 15, 22, 4, 2, 7, 3 }
```

NOTE: Now that our programs have more functions than just the `main()` function, the use of the function headers becomes meaningful and required. Make sure that all of your functions have function headers preceding them that document the purpose of the functions, and the input parameters and return value of the function.

Assignment Submission

An eCollege dropbox has been created for this assignment. You should upload your version of the assignment to the eCollege dropbox named `Assg 12 Positive Ints Library` created for this submission. Work submitted by the due date will be considered for evaluation.

Requirements and Grading Rubrics

Program Execution, Output and Functional Requirements

1. Your program must compile, run and produce some sort of output to be graded. 0 if not satisfied.
2. 20+ pts. For the correct implementation of new functionality in `postintcmp()` function.
3. 20+ pts. For the correct implementation of the `postintcpy()` function.

4. 25+ pts. For correctly implementing the `postintfind()` function.

5. 25+ pts. For correctly implementing the `postintcat()` function.

6. 10+ pts. For demonstrating your functions adequately in your `main()` function.

Program Style

Your programs must conform to the style and formatting guidelines given for this course. The following is a list of the guidelines that are required for the assignment to be submitted this week.

1. The file header for the file with your name and program information and the function header for your main function must be present, and filled out correctly.

2. A function header must be present for all functions you define. You must document the purpose, input parameters and return values of all functions. Your function headers must be formatted exactly as shown in the style guidelines for the class.

3. You must indent your code correctly and have no embedded tabs in your source code. (Don't forget about the Visual Studio Format Selection command).

4. You must not have any statements that are hacks in order to keep your terminal from closing when your program exits (e.g. no calls to system()).

5. You must have a single space before and after each binary operator.

6. You must have a single blank line after the end of your declaration of variables at the top of a function, before the first code statement.

7. You must have a single blank space after , and ; operators used as a separator in lists of variables, parameters or other control structures.

8. You must have opening `{` and closing `}` for control statement blocks on their own line, indented correctly for the level of the control statement block.

9. All control statement blocks (if, for, while, etc.) must have `{ }` enclosing them, even when they are not strictly necessary (when there is only 1 statement in the block).

10. You should attempt to use meaningful variable and function names in your program, for program clarity. Of course, when required, you must name functions, parameters and variables as specified in the assignments. Variable and function names must conform to correct `camelCaseNameingConvention` .

11. Put the `*` for pointer variable declarations next to the type declaration, with no space between the type and the `*`. Also please follow the convention of using `Ptr` at the end of names for pointer variables.

Failure to conform to any of these formatting and programming practice guidelines for this assignment will result in at least 1/3 of the points (33) for the assignment being removed for each guideline that is not followed (up to 3 before getting a 0 for the assignment). Failure to follow other class/textbook programming guidelines may result in a loss of points, especially for those programming practices given in our Deitel textbook that have been in our required reading so far.