Lab 12:

CSci 515 Spring 2015

<2015-04-21 Tue>

Dates:

Due: In Lab, Wednesday April 22, by 4:05 pm (lab end time)

Objectives

- Understand the relationship between arrays and pointers in C
- Use pointers to an array of items, and process them.
- Understand better C-string processing by implementing sentinel based processing functions.

Description

In this lab we will implement a similar concept to passing arrays of characters for string processing by defining an array of (positive) integer type. Arrays of positive integers can be of varying length. Since arrays of positive integers must always contain values $v[i] >= 0$ we can use the value of `-1` as a sentinel character. Thus we will not pass array sizes to function that process arrays of positive integers, instead all such function will expect that a `-1` terminal sentinel will be set as the final value for such arrays. In this lab you will implement functions to accept a pointer to an array of positive integers, that implement the calculating the length of such arrays and comparing if two arrays are equal.

Perform the following tasks:

1. Create a function called `posintprint()`. This function should take a pointer to an array of integers as its first and only argument. This function is a void function, it doesn't display any result. However,

this function should cause the values of the list of positive integers provided to be displayed to standard output as its result. See the example output for how this output should be displayed. (As extra credit, try not to have the last unnecessary ',' printed out).

2. Create a function called `posintlen()`. This function should take a pointer to an array of integers as its first and only argument. This function will expect that the array is terminated with the `-1` sentinel character correctly. The function should return an integer value, which is the number of positive integer values that is contained in the array.

3. Create a function called `posintcmp()`. This function should take a pointer to an array of (positive, `-1` terminated) integers as its first and second parameters. This function should return an integer. As with the `strcmp()` function, this function should compare all corresponding elements of the two arrays of positive integers provided as parameters. It will return a 0 if all of the elements in both arrays are 0 (you can assume for the lab for today that the two items will be of the same length), and it will return -1 or +1 as appropriate, depending on if the first array is less than or greater than the second at the first position that differs between the two input parameters.

4. In your `main()` function make sure you demonstrate your code and perform the following tests. Test that your `posintlen()` function works for lists of length 5 and of length 0. Test that your `posintcmp()` function returns 0 for 2 equal lists, and +1 and -1 for two unequal lists as appropriate.

Example output is shown below (value between the { and } are being displayed by the `posintprint()` function for different lists of positive integers):

```
List 1: { 5, 3, 8, 2, 7, }
Length of List 1: 5

List 2: { }
Length of List 2: 0

List 3: { 4, 2, 9, }
List 4: { 4, 2, 9, }
Result of posintcmp(list3, list4): 0
```

```
List 3: { 4, 2, 9, }
List 5: { 4, 2, 7, }
Result of posintcmp(list3, list5): 1
Result of posintcmp(list5, list3): -1
```

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.

Lab Submission

An eCollege dropbox has been created for this lab. You should upload your version of the lab by the end of lab time to the eCollege dropbox named `Lab 12 Arrays of Positive Ints`. Work submitted by the end of lab will be considered, but after the lab ends you may no longer submit work, so make sure you submit your best effort by the lab end time in order to receive credit.

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. 35+ pts. Your `posintlen()` function must work correctly and take a pointer to an array of integers as specified.
3. 40+ pts. Your `posintcmp()` function works correctly and takes two pointers to arrays of positive integers as specified.
4. 25+ pts. You main function demonstrates correctly invoking the functions and displays the required output from doing so.

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 lab 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 lab 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.