EECS 280 Lab 04: Arrays and Pointers

Due Sunday, 7 February 2016, 8:00pm

In this lab, you will review the basics of pointers in C/C++ and practice using them to traverse and manipulate arrays.

This lab covers material from these lectures:

- 06 Arrays and Pointers
- 07 Array Traversal

Overview

Requirements

You may work on this lab either individually or in groups of 2-3. Include your name(s) in the comments at the top of the file. Submit the files below on CTools. You do not need to turn in any other files. If you work in a group, each person must submit a copy in order to receive credit.

Files to submit:

- `lab04.cpp`

Completion Criteria:

To pass this lab you must complete task 1. Task 2 is optional.
This checklist will give you an idea of what we look for when grading for completion:

✓ (Task 1) Implement `printRowIndex`. You must use traversal by index.
✓ (Task 1) Implement `printRowPtr`. You must use traversal by pointer.
✓ (Task 2) (Optional) Implement `slideRight`. You must use traversal by pointer.
✓ (Task 2) (Optional) Implement `flipHorizontal`. You must use traversal by pointer.
✓ (Task 2) (Optional) Implement `invertColors`. You must use traversal by pointer.
For each of the above, your implementation does not have to be perfect, but should be reasonably close to correct.

Task 0 - Preliminaries

The Files

We have provided a skeleton file in which you should write your code. If you have a terminal open, the following command will automatically download it from the eecs280 Google Drive repository to your current directory:

```
$ wget goo.gl/UKI8I0 -O - | tar xzk
```

In case you are working locally and want to manually download the files, they are also attached to the CTools assignment and available on the course Google Drive Repository (see link in header).

Here's a brief summary of the files included in this lab. Files you need to turn in are shown with a red background.

`lab04.cpp`	Contains function stubs for `printRowIndex`, `printRowPtr`, `slideRight, flipHorizontal and invertColors`. This file includes the `main` function and testing code.

Testing Code

The starter code should "work" out of the box, so make sure you are able to compile and run it. The code may be missing some pieces, contain some bugs, or crash when you run it, but you'll fix each throughout the course of the lab.

Introduction

Our goal in this lab is to write a function that can print simplified ASCII art pictures. In a nutshell, the idea is that we can arrange a grid of characters in a pattern so that the end result looks like an image. For now, we'll only consider a very simple subset that uses only two characters: "#" for an "on" pixel and " "(a space) for an "off" pixel. Here's an example. (Yes, it is beautiful.)

```
 0 1 2 3 4 5 6 7 8 9
0    # # # # #
1  #           #
2#   #       #   #
3#               #
4#       #       #
5#               #
6#           #   #
7#     # # #     #
8  #           #
9    # # # # #
```

Note the way in which the grid is numbered. There are 10 rows and 10 columns. The origin is in the top left, and we use indices starting at 0. We'll use arrays to represent this grid in our program. Specifically, each row will be an array of 10 integers. Each element will either be 0 (corresponding to " ") or 1 (corresponding to "#").

So to initialize the first (0th) row above we would write:

int row0[10] = {0, 0, 1, 1, 1, 1, 1, 0, 0, 0};

Since it would be cumbersome to have a different variable for each row, we'll instead store the rows themselves as part of a 2D array. We would write:

int grid[10][10];

grid[0] = {0, 0, 1, 1, 1, 1, 1, 0, 0, 0};

Traversing Arrays

There are two basic ways in which we can traverse an array: traversal by index and traversal by pointer. Each has advantages and disadvantages in particular situations. Throughout this lab and the course in general, we may ask you to use one over the other. We will contrast them below with two snippets of code that add up all the elements in an array.

Traversal by Index

```
int arr[5] = {1,2,3,4,5};
int sum = 0;
for(int i = 0; i < 5; ++i){
  sum += arr[i]; // or sum += *(arr+i)
}
```

Traversal by Pointer

```
int arr[5] = {1,2,3,4,5};
int sum = 0;
for(int *ptr = arr; ptr < arr + 5; ++ptr){
  sum += *ptr;
}
```

The biggest difference is whether the variable that controls the loop is an index (i.e. an integer) or a pointer. There are a few other things worth noting:

	Traversal by Index	**Traversal by Pointer**
Initialization	The index starts at 0.	The pointer starts at the address of the first element.
Loop condition	Loop while the index is < length.	Loop while the pointer has not reached address `arr + length`, which is past the end of the array.
Increment	Increase index by 1.	Move the pointer forward by 1.
Element access	Array indexing - use index as offset from start of array and then dereference. Remember that `arr[i] == *(arr+i)`.	The pointer is already pointing to the current element, so just dereference it (i.e. `*ptr`).

Task 1 - Printing an Array

Your first task is to write two versions of a function to print out a single row of the grid. In the first version, you should use traversal by index to iterate through the array. In the second version, you should use traversal by pointer. Find the printRowIndexand printRowPtrfunction in lab04.cpp and complete their implementations.

```
// REQUIRES: there are at least size elements in row[]

// EFFECTS: For each element of row (in sequence):

//          if that element is a 0, prints out "  " to cout

//          if that element is a 1, prints out "# " to cout

//          (note the extra space after each)

//          DOES NOT print any newlines

void printRowIndex(const int row[], int size){

     cout << " "; // TASK 1  REPLACE WITH YOUR CODE

     // NOTE: You should use traversal by index in printRowIndex

}


// REQUIRES: row points to an array with at least size elements

// EFFECTS: For each element of row (in sequence):

//          if that element is a 0, prints out "  " to cout

//          if that element is a 1, prints out "# " to cout

//          DOES NOT print any newlines

void printRowPtr(const int *row, int size){

     cout << " "; // TASK 1  REPLACE WITH YOUR CODE

     // NOTE: You should use traversal by pointer in printRowPtr

}
```

Task 2 - Array Transformations (Optional)

Now, let's implement some functions that apply transformations to an array. In these functions, you must use **traversal by pointer**.

- **Slide Right** - All elements are shifted right by 1 unit. The last element wraps around to the beginning.
- **Flip Horizontal** - The order of elements in the array is reversed.
- **Invert Colors** - All elements that are 1 are switched to 0; all that are 0 are switched to 1

These transformations can all be applied row by row such that it makes sense for us to just write functions to operate on a single row at a time. Then we can use the `applyTransformation` function already in lab04.cpp to apply it to the entire grid.

You will find function stubs for each of the transformations in `lab04.cpp`. Fill in the implementation for each. The main function already in `lab04.cpp` contains code to test these as well, but you will need to uncomment it.