Carleton University
Department of Systems and Computer Engineering
SYSC 2006 - Foundations of Imperative Programming - Fall 2015

Lab 5 - Arrays and Pointers

Attendance/Demo

To receive credit for this lab, you must demonstrate your solutions to the exercises. **Also, you must submit your work to cuLearn**. (Instructions are provided in the *Wrap-up* section at the end of this handout.)

When you have finished all the exercises, call a TA, who will review the code you wrote. For those who don't finish early, a TA will ask you to demonstrate whatever code you've completed, starting about 30 minutes before the end of the lab period. **Any unfinished exercises should be treated as "homework"; complete these on your own time, before your next lab.**

General Requirements

For those students who already know C or C++: when writing the code for the exercises, do not use `struct`s. They aren't necessary for this lab.

None of the functions you write should perform console input; i.e., contain `scanf` statements. None of your functions should produce console output; i.e., contain `printf` statements.

You have been provided with file main.c. This file contains incomplete implementations of three functions you have to design and code. (There's also an incomplete implementation of an extra-practice exercise). It also contains a *test harness* (functions that will test your code, and a `main` function that calls these test functions). **Do not modify `main` or any of the test functions.**

Getting Started

Step 1

Launch Pelles C and create a new project named arrays_pointers (all letters are lowercase, with underscores separating the words). If you're using one of our lab computers, the project type must be Win 64 Console program (EXE). (Although the 64-bit edition of Pelles C can build 32-bit programs, you may run into difficulties if you attempt to use the debugger to debug 32-bit programs.) If you're using your own computer, the project type should be Win 64 Console program (EXE) or Win32 Console program (EXE), depending on whether you installed the 64-bit or 32-bit edition of Pelles C. After creating the project, you should have a folder named arrays_pointers. Check this. If you do not have a project folder named arrays_pointers, close this project and repeat Step 1.

Step 2

Download files main.c and sput.h from cuLearn. Move these files into your arrays_pointers folder.

Step 3

You must also add main.c to your project. To do this, select Project > Add files to project... from the menu bar. In the dialogue box, select main.c, then click Open. An icon labelled main.c will appear in the Pelles C project window.

You don't need to add sput.h to the project. Pelles C will do this after you've added main.c.

Step 4

Build the project. It should build without any compilation or linking errors.

Step 5

Execute the project. The test harness will report several errors as it runs, which is what we'd expect, because you haven't started working on the functions the harness tests.

Step 6

Open main.c in the editor. Design and code the functions described in Exercises 1 through 3.

Debugging Tip

I recommend that you write a function that prints the first *n* integers in an array:

```
void print_array(int arr[], int n)
```

The output can be as simple as: `1 2 3 4 5 6 7`

or the formatting can be more fancy; e.g., `{1, 2, 3, 4, 5, 6, 7}`

If one of your functions doesn't work, add calls to `print_array` at appropriate places in your function; e.g., before the `for` or `while` loop that traverses the array; in the loop body, after an array element is modified; and after the loop.

Exercise 1

Write a function that reverses the values in an array containing *n* integers. The function prototype is:

```
void reverse(int arr[], int n);
```

Note: your function should assume that `n` is positive; i.e., it should not check whether `n` is passed a positive or negative value.

As an example, suppose the function is called this way:

```
int numbers[] = {1, 2, 3, 4, 5, 6, 7};
reverse(numbers, sizeof(numbers) / sizeof(numbers[0]);
```

When the function returns, array `numbers` will be: `{7, 6, 5, 4, 3, 2, 1}`.

Your `reverse` function must call the `swap` function that was presented in one of the lectures. (A copy of this function is in main.c). Only the `swap` function is permitted to modify individual elements in the array. In other words, your function cannot have any statements of the form:

```
arr[i] = expression;
```

Similarly, if you decide to use the pointer-plus-offset or walking-pointer approaches to traverse the array, your function cannot have any statements of the form:

```
*(pa + i) = expression;
```

or

```
*pa = expression;
```

Hint: your function shouldn't consider arrays with an even number of elements and arrays with an odd number of elements as separate, distinct cases. There's no need to code something like this:

```
if (array has an even number of elements) {
    reverse the array
} else {         // array has an odd number of elements
    reverse the array
}
```

Build the project, correcting any compilation errors, then execute the project. The test harness will run. Look at the console output, and verify that your function passes all the tests in test suite #1 before you start Exercise 2.

Exercise 2

Write a function that "rotates" the *n* integers in an array one position to the left. For example, the function will change the array `{6, 2, 5, 3}` to `{2, 5, 3, 6}`. The function prototype is:

```
void rotate_left(int *arr, int n);
```

Note: your function should assume that `n` is positive; i.e., it should not check whether `n` is passed a positive or negative value.

Do not use the indexing (`[]`) operator. Instead, use "pointer-plus-offset" notation to access the array elements. (See the lecture slides on pointers and arrays for more information.)

Build the project, correcting any compilation errors, then execute the project. The test harness will run. Look at the console output, and verify that your function passes all the tests in test suite #2 before you start Exercise 3.

Exercise 3

Write a function that is passed an array of *n* integers. For each multiple of 10 in the given array, change all the values following it to be that multiple of 10, until encountering another multiple of 10. For example, the function will change the array `{2, 10, 3, 4, 20, 5}` to `{2, 10, 10, 10, 20, 20}`. The function prototype is:

```
void ten_run(int *arr, int n);
```

Note: your function should assume that `n` is positive; i.e., it should not check whether `n` is passed a positive or negative value.

Do not use the indexing (`[]`) operator. Instead, use the "walking-pointer" approach to traverse the array and access the individual elements. (See the lecture slides on pointers and arrays for more information.)

Build the project, correcting any compilation errors, then execute the project. The test harness will run. Look at the console output, and verify that your function passes all the tests in test suite #3.

Note: there is an extra-practice exercise after the wrap-up instructions, to help you prepare for the midterm exam.

Wrap-up

1. Remember to have a TA review your solutions to the exercises, assign a grade (Satisfactory, Marginal or Unsatisfactory) and have you initial the demo/sign-out sheet.

2. The next thing you'll do is package the project in a ZIP file (compressed folder) named arrays_pointers.zip. To do this:

   2.1. From the menu bar, select Project > ZIP Files... A Save As dialog box will appear. If you named your Pelles C project arrays_pointers, the zip file will be named arrays_pointers.zip by default; otherwise, you'll have to edit the File name: field and rename the file to arrays_pointers before you save it. **Do not use any other name for your zip file** (e.g., lab5.zip, my_project.zip, etc.).

   2.2. Click Save. Pelles C will create a compressed (zipped) folder, which will contain copies of the the source code and several other files associated with the project. (The original files will not be removed). The compressed folder will be stored in your project folder (i.e., folder arrays_pointers).

3. Before you leave the lab, log in to cuLearn and submit arrays_pointers.zip. To do this:

   3.1. Click the Submit Lab 4 link. A page containing instructions and your submission status will be displayed. After you've read the instructions, click the Add submission button. A page containing a File submissions box will appear. Drag arrays_pointers.zip to the File submissions box. **Do not submit another type of file (e.g., a Pelles C .ppj file, a RAR file, a .txt file, etc.)**

   3.2. After the icon for the file appears in the box, click the Save changes button. At this point, the submission status of your file is "Draft (not submitted)". If you're ready to finish submitting the file, jump to Step 3.4. If you aren't ready to do this; for example, you want to do some more work on the project and resubmit it later, you can leave the file with "draft" submission status. When you're ready to submit the final version, you can replace or delete your "draft" file submission by following the instructions in Step 3.3, then finish the submission process by following the instructions in Step 3.4.

   3.3. You can replace or delete the file by clicking the Edit my submission button. The page containing the File submissions box will appear.

      3.3.1. To overwrite a file you previously submitted with a file having the same name, drag another copy of the file to the File submissions box, then click the Overwrite button when you are told the file exists ("There is already a file called..."). After the icon for the file reappears in the box, click the Save changes button.

3.3.2. To delete a file you previously submitted, click its icon. A dialogue box will appear. Click the Delete button., then click the OK button when you are asked, "Are you sure you want to delete this file?" After the icon for the file disappears, click the Save changes button.

3.4. Once you're sure that you don't want to make any changes, click the Submit assignment button. A Submit assignment page will be displayed containing the message, "Are you sure you want to submit your work for grading? You will not be able to make any more changes." Click the Continue button to confirm that you are ready to submit your lab work. This will change the submission status to "Submitted for grading".

Extra-Practice Exercise

This exercise is more challenging than Exercises 1-3. A correct solution will typically require between 15 and 20 lines of code. (Lines containing only a `}` are counted as one line of code. Comments are not counted as lines of code.)

Write a function that removes all the 10's from an array of *n* integers. The remaining elements should be shifted left towards the start of the array as required, and the "empty" spaces at the end of the array should be set to 0. For example, the function will change the array `{1, 10, 10, 2, 10, 3}` to `{1, 2, 3, 0, 0, 0}`. The function prototype is:

```
    void without_tens(int *arr, int n);
```

Note: your function should assume that `n` is positive; i.e., it should not check whether `n` is passed a positive or negative value.

You can use the indexing (`[]`) operator or pointer dereferencing (pointer-plus-offset or walking-pointer) to access individual array elements. Suggestion: write three versions of the function, one using the indexing operator approach, another using pointer-plus-offset approach, and the third using the walking-pointer approach. Which version is easier to understand?

Hint: the "obvious" solution uses nested loops (the outer loop searches the array for the next 10 to be removed, and the inner loop shifts subsequent elements to the left), so you might want to start by developing that algorithm. If you want a challenge, develop a solution that does not use nested loops.

Build the project, correcting any compilation errors, then execute the project. The test harness will run. Look at the console output, and verify that your function passes all the tests in test suite #4.

Acknowledgments

Some of these exercises were adapted from Java programming problems developed by Nick Parlante.