Carleton University
Department of Systems and Computer Engineering
SYSC 2006 - Foundations of Imperative Programming - Winter 2014

Lab 4

Attendance/Demo

To receive credit for this lab, you must make a reasonable effort to complete the exercises and demonstrate the code you complete.

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. Finish any exercises that you don't complete by the end of the lab on your own time. Also, you must submit your lab work to cuLearn by the end of the lab period. (Instructions are provided in the *Wrap Up* section at the end of this handout.)

General Requirements

For those students who already know C or C++: when writing the code for this lab's exercises, do not use structs. 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 five functions you have to design and code. 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.**

Instructions

Step 1

Create a new folder named Lab 4.

Step 2

Create a new project named pointers inside your Lab 4 folder. The project type must be Win32 Console program (EXE). After creating the project, you should have a folder named pointers inside your Lab 4 folder (check this).

Step 3

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

Step 4

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 5

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

Step 6

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 7

Open main.c in the editor. Design and code the functions described in Exercises 1 through 5.

Exercise 1

Here's a simple warmup-up exercise that requires you to write a function that has pointer parameters.

A cube is a geometric solid consisting of six square faces that meet each other at right angles. Write a function that calculates the surface area and the volume of a cube. The function prototype is:

```
void cube_area_volume(double len, double *area, double *volume);
```

Parameter `len` is the length of the cube's sides. Assume that `len` will always be positive (your function does not have to check this.) Parameters `area` and `volume` point to the variables where the function will store the area and volume that the function calculates.

Suppose we have the declarations for two variables, `a` and `v`:

```
double a;
double v;
```

To calculate the surface area and volume of a cube whose sides have length 2, the function is called this way:

```
cube_area_volume(2, &a, &v);
```

When the function returns, the calculated area and volume will be in variables `a` and `v`, respectively.

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 the test suite before you start Exercise 2.

Exercise 2

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, if the function is called this way:

```
int a[] = {1, 2, 3, 4, 5, 6, 7};
reverse(a, 7);
```

when the function returns the array 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). Your function cannot have any statements of the form:

```
arr[i] = variable;
```

or:

```
variable = arr[i];
```

In other words, only the `swap` function is permitted to access individual elements in the array.

Hint: your function does not need to consider arrays with an even number of elements and arrays with an odd number of elements as separate cases. One algorithm can handle both cases.

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 the test suite before you start Exercise 3.

Exercise 3

Write a function that "rotates" the *n* integers in an array one position to the left. If the function is passed the array `{6, 2, 5, 3}`, the function changes the array 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 the test suite before you start Exercise 4.

Exercise 4

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. So, 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 the test suite before you

start Exercise 5.

Exercise 5

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. So, 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 to access individual array elements.

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 the test suite.

Wrap-up

1. Remember to have a TA review and grade your solutions to the exercises before you leave the lab.

2. The next thing you'll do is package the project in a ZIP file (compressed folder). From the menu bar, select Project > ZIP Files... A Save As dialog box will appear. Click Save. Pelles C will create a compressed (zipped) folder named pointers.zip, 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 pointers).

3. Log in to cuLearn, click the Submit Lab 4 link and submit pointers.zip. After you click the Add submission button, drag the file to the File submissions box. After the icon for the file appears in the box, click the Save changes button. At this point, the submission status for your file is "Draft (not submitted)". You can resubmit the file by clicking the Edit my submission button. Once you're sure that you don't want to make any changes, click the Submit assignment button. This will change the submission status to "Submitted for grading". **Note: after you've clicked the Submit assignment button, you cannot resubmit the file.**

Acknowledgments

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