CS270 Recitation 11
"C Pointer Exercise"

Goals

To improve your understanding of C pointers, including arrays and strings, via a practice session.

The Assignment

Make a subdirectory called R11 for the recitation assignment, all files should reside in this subdirectory. Download the pointers.c source file and associated Makefile as a starting point for the exercise. Make sure the program compiles and runs in the R11 directory:

```
$ make
$ ./r11
```

Exercise 1: Pointer Basics

The exercise1() function has the code for the first exercise. The function 1) declares several different types of variables, 2) initializes the variables, 3) declares a pointer for each variable, and 4) initializes the pointer to point at the variable. The values and address of each variable are printed.

STEP 1: Add code that prints out the pointer and the address of the variable (using the & symbol), and verify that they are the same.

STEP 2: Add code that changes the value of the variable via the pointer (using the * symbol), then print out the pointers and values again to test your code.

Conclusions:

- **The address of a variable and a pointer to the variable are identical.**
- **Pointers can be used to change the value of variables.**

Exercise 2: Pointer Arguments

The exercise2() function has the code for the second exercise. The code defines two subfunctions called passByValue() and passByReference(), which pass their arguments by value and references.

STEP 1: Add code to call the passByValue() function with the local variables. Check the values printed out to make sure that they don't change after the call.

STEP 2: Add code to call the passByReference() function with the local variables. Check the values printed out to make sure that they do change after the call.

Conclusions:

- **Arguments can be values or pointers.**
- **The value of a variable can be changed by a function only when passed by reference.**

Exercise 3: Arrays and Pointers

The exercise3() function has the code for the third exercise. The function declares a static array, then it allocates dynamic memory for another array, and both are initialized.

STEP 1: Add code to print out the static array, using pointer access instead of array subscripts, e.g. use *(intArray+0) to access the first element, *(intArray+1) to access the second element, etc.

STEP 2: Add code to print out the dynamic array, using array subscripts instead of pointer access, e.g. use intPointer[0] to access the first element, intPointer[1] to access the second element, etc.

Question: Why is it intArray+1 instead of intArray+4, isn't an integer four bytes?
Answer: **Pointer math accounts for size of objects being pointed at.**

Conclusions:

- **Arrays can be allocated statically or dynamically.**
- **Arrays and pointers are virtually interchangeable in C.**

Exercise 4: String and Pointers

The exercise4() function has the code for the fourth exercise. The code defines three strings by using three methods: an initializer, static allocation, and dynamic allocation.

STEP 1: Add code to print out these strings using %s and the name

STEP 2: Add code to print out the seventh character using %c and array access, e.g. string1[6].

Conclusions:

- **You can allocate strings different ways, but they're all the same.**
- **Character pointers and arrays of characters are identical, they're both strings.**

Exercise 5: Static Versus Dynamic

The exercise5() function has the code for the fifth exercise. The code allocates dynamic and static arrays of several data types.

STEP 1: Add code to print out the pointers to the static and dynamic arrays, no special syntax is required, just use the array or pointer name, since both are just pointers! The allocations should have very different addresses.

Conclusions:

- **Memory can be allocated dynamically or statically in C.**
- **Local static allocations are on the stack, dynamic allocations are on heap.**