**CS2060 Fall 2017**

**Exam #2 Review**

**Chapter 5 – C Functions**

* Understand the reason modular programming is important in programming

Reusable and easier to understand

* Know how to use these 2 basic math functions - sqrt, pow

sqrt (double), pow (number, power)

* Know how to define and create a function, so understand:
  + Function prototypes
  + Function invocation
  + Function definition
* Understand the following terms and their role in creating and using functions:
  + Return type
  + Formal parameters

(Parameter) found in function definition

* + Actual parameters

(Argument) found in function call

* Understand the difference between ***formal*** and ***actual*** parameters
* Understand the difference between a ***value-returning*** function and a ***void*** function
* Understand how to pass values to a function and how to get a value back
* Understand automatic conversions:
  + Know that conversions occur when argument types do not correspond to parameter types

Converted before being passed

* + Know what kinds of issues automatic conversions can cause

Incorrect results

* Understand the scope of variables:
  + Know how to determine the scope of variables
  + What local variables are
  + Understand how the call stack works and how this relates to scope of variables

Variables only exist as long as function is active

* Understand what ***pass by value*** means and the impact on variables when a function is called
* Know what recursion is on a high-level, you will NOT be asked to write a recursive function
* Know how to write a function if given a simple task
  + For example, write a function that computes the sum of the numbers from 1 to N
* Be able to explain code, trace code, and write code snippets with functions

**Chapter 6 – C Arrays**

* Know how to declare, create, and initialize an array (one or two dimensional)

Int array[]={1,2,3}, int array2[][2] = { {0,0}, {1,2}}

* Know how to access elements within an array (one or two dimensional)
* Understand initialization of arrays
  + What is in array when you don’t initialize the array

Garbage

* + Understand initializer lists (i.e. how they work, what if fewer items than array elements, etc.)

Rest 0 if not enough

* How to manipulate elements within an array (i.e. compute sum, find largest, display elements, etc.)
* Understand what happens with arrays when
  + Code accesses elements in an array outside the array bounds

C allows this, can have garbage

* + There is an off-by-one situation (mostly issue when forgetting arrays are zero-based)
* Understand character arrays
  + Know how to create and initialize strings using character arrays

Char string[] = “first”;

* + Understand the importance of the null character in strings

All strings have null char on end ‘\0’

* + How to read and display strings

Prinf(“%s”, string);

* Understand and know how to ***pass arrays to functions***
  + Understand what ***pass by reference*** means and the impact when passing an array to a function
  + Know how to pass an entire array or a single element in the array to functions

Pass array name only for whole array

* + Understand when to use ***const*** qualifier on an array in a parameter list

Don’t want function to change array

* Understand the bubble sort on a high level
  + What does it do
  + How efficient it is
* Know how to perform a linear search for a key value
* Understand the concept of how the binary search works on a high level
  + What does it do

Small values bubble up, large values sink down

* + How efficient it is

Not very efficient, O(n^2)

* + When it should or should not be used

Small arrays

* Know when one search is better over the other

Linear avg search half (small or unsorted), Binary eliminates half array(sorted) each comparison

* Be able to explain code, trace code, and write code snippets with arrays

**Chapter 7 – C Pointers**

* Understand what a pointer variable is
  + How pointers are different from other variables

Memory addresses

* + What is the purpose of pointers

Pass by reference, pass functions between functions

* Know how to create a pointer and initialize it

Int \*intPtr = &intValue;

* Know how to use the pointer operators & and \*
  + Know what does each operator does

&-address, \*-dereference pointer (get value pointer points to)

* + Know when do you use one over the other
  + Understand code that contains these operators
* Understand and know how to ***pass pointers to functions***
  + Understand what ***pass by reference*** means and the impact when passing a pointer to a function
  + Understand and know what needs to be in argument and parameter lists when passing pointers

Parameter list- int \*ptr, argument list- &int

* Understand the 4 different cases of using the **const** qualifier with pointers
  + Non-constant pointer to non-constant data

Both can be edited

* + Constant pointer to non-constant data

Pointer doesn’t change, data can

* + Non-constant pointer to constant data

Pointer can change, data cannot

* + Constant pointer to constant data

Neither can change

* Know how to use the ***sizeof*** operator

Size in bytes, size\_t

* Know how to use pointers in expressions and how to perform pointer arithmetic
  + Comparing pointers

Doesn’t make sense, typically compare to null

* + Incrementing and decrementing pointers

Int \*pointer+1 is incrementing by 4 bytes (size of int), ++, --

* + Adding and subtracting values from pointers
  + Subtracting one pointer from anther

Number of elements in between

* Understand what a void pointer is

No type, can’t be dereferenced

* Understand the relationship between arrays and pointers
* Understand how to create, access, manipulate an array of pointers
* Understand pointers to functions at a high level
* Be able to explain code, trace code, and write code snippets with pointers

**Overall**

* Be prepared to write small snippets for:
* Value returning or void functions
  + Passing and returning scalar types (int, float, etc.)
  + Passing and returning arrays
  + Passing and returning pointers
* Manipulating arrays
  + One or two dimensional
* Manipulating pointers