CS 162, Lecture 2: Pointer, Array and Struct Review

6 April 2018
Pointers

• Pointers == memory addresses

• Variable declaration: int a = 5;
  • Creates a variable on the stack of size int with the value 5.

• Pointer declaration: int* b = &a;
  • Creates a pointer variable on the stack which can hold an address of an int and sets the value of the pointer (the address the pointer points to) to the address of a

• Dereferencing Pointer: cout << *b << endl;
  • Will print the value stored at the address which b points to
Pointers and Functions

• void func(int a, int b);
  • Prototype indicating that void func is expecting two parameters of type int to be passed by value
  • Recall pass by value copies the value being passed into the formal parameters which are scoped to this function, any changes made to the values in the function will not reflect outside this scope

• void func(int* a, int b);
  • Prototype indicating that void func is expecting two parameters, one of type int* (an address of an int) and one of type int.
  • Since a is of type int* (an address) it will have to be dereferenced throughout the function if the value stored at that address is to be used. a may also receive a new address in the function. Changes made to this variable will persist beyond the scope of this function
Demo Pointers
Arrays

• An array is of one data type and its memory is stored contiguously
• Static Arrays: created on the stack and are of a fixed size
• Dynamic Arrays: created on the heap and their size may change during runtime
• Arrays may be of one or more dimensions
Array Demo
Structs

• User defined objects
• Container which holds many variables of different types

```c
struct my_obj {
    string name;
    int some_stat;
    float* some_list_of_stats;
};
```

• Can be used as any other data type with some extra features
  • Access members using the dot operator
    • Ex: my_obj.name = "Obj1";
  • If the struct needs to be dereferenced use the arrow
    • Ex: my_obj->name = "Obj1";
Struct Demo