CS 162: Reviewing the Fundamentals

Pointers, Arrays, and Stack vs Heap

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

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

```
int stack_array[10];
```

- Dynamic Arrays: created on the heap and their size may change during runtime

```
int *heap_array = new int[10];
```

- Arrays may be of one or more dimensions

```
int stack_array_2d[5][7];
```

Pointers and Functions

- void my_func(int a, int b);
  - Prototype indicating that this void function 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 my_func(int* a, int b);
  - Prototype indicating that void function 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

Structs

- User defined objects
- Container which holds many variables of different types
    - Very useful!
- Can be used as any other data type with some extra features

How to define a struct

```
// definition of a book struct
struct book {
      int pages;
      unsigned int pub_date;
      string title; // a string inside the struct
      int num_authors;
      string* authors; // a pointer to a string
};

// declare a book struct
book text_book;
```

Working with structs

- Can use the same way as any other type
- The . operator allows us to access the member variables

```
book bookshelf[10];
for (int i = 0; i < 10; i++) {
    bookshelf[i].num_pages = 100;
    bookshelf[i].title = “Place holder”;
    bookshelf[i].authors = new string[2];
}
```

Using pointers with structs

```
book bk1; // statically allocated
book* ptr1 = &bk1;

// dereference the pointer and access the data member
(*ptr1).title = “Old Yeller”;

// a shortcut to dereference the struct
// the -> operator
ptr1->title = “Old Yeller (abridged)”;
ptr1->num_pages = 196;

// this works for objects on the heap as well
book* ptr2 = new book; // dynamically allocated
ptr2->title = “Charlotte’s Web”;
```