CS 261 – Data Structures

C Programming Basics Review
Why C?

C is a straightforward procedural language that makes it easier to focus on important concepts.

Avoid OOP Baggage

- Classes
- Inheritance
- Polymorphism
- Function overloading

However....

- No garbage collection
- No reference types

Memory Management
Pointers
Every C Program has a main

```c
int main (int argc, char **argv) {
    ...
}
```

Main kicks off execution and can call other **functions**

Command Line Arguments:

- `argc = 3`
- `argv[0] = computeRectangleArea`
- `argv[1] = 10`
- `argv[2] = 22`
Function Definitions

Functions look a lot like methods you are used to in Java, but are not part of a class:

\[
\text{return-type function-name(parameters)} \{ \\
\text{variable-declarations;} \\
\text{function-body;} \\
\}
\]

Example — return sum of elements of an integer array:

\[
\text{long arrSum(int arr[], unsigned int n)} \{ \\
\text{unsigned int } \text{i;} \\
\text{long sum }= 0; \\
\text{for (i }= \text{ 0; } \text{i }< \text{ n; i++) } \{ \\
\text{sum }+= \text{ arr[i];} \\
\} \\
\text{return sum;} \\
\}
\]

Need to pass size of array (not included in \text{arr}).
Structures (user defined types)

Structures are like classes that have only public data fields and no methods:

```c
struct Gate {
    int       type;  /* Type of gate. */
    struct Gate *left; /* Left input. */
    struct Gate *right; /* Right input. */
};
```

A diagram of a simple gate with inputs A and B and output Y.
Accessing Struct Fields

Access to struct fields uses the same dot notation you are used to:

```c
struct Gate gate;
gate.type = 3;
```

(but often combined with pointers ...more on this later!)
Object Oriented vs. Procedural

In OOP (e.g. Java), we define classes with methods and call methods ‘on’ class instances

    student s = new Student();
    s.print();

In C, we define functions and in order to use a structure with that function, we must pass the structure into the function

    void printStudent(struct Student myStudent)
    {
        /* Code to print a single student struct*/
    }
    ...
    struct Student s;
    /*fill s*/
    printStudent(s)
Scope (simplified)

Global

• variables declared outside of any function (use sparingly)

Local

• variables declared inside of function

```c
double avg;  /* Global variable: can access in any function. */

void arrAvg(int arr[], unsigned int n) {
    unsigned int i;  /* Local variables: access only within function. */
    long sum = 0;

    for (i = 0; i < n; i++) sum += arr[i];
    avg = (double)sum / n;
}
```
And much, much more...

Types
- char
- int
- float
- double

Comments
- /* Ansi C (C89) */
- // Post C89

Control
- if-else statements
- if-else if statements (for multiway decisions)
- switch statements
- while loops
- for loops
- do-while loops

Get a good reference