Overview of C

- Most of C programming is like working in Java, because the syntax is almost the same. The biggest difference is that in C there are no classes. The simplified structure of a C program might remind you of Python.

- C is a great language for writing small to medium sized programs that need to run fast. Those are the kind of programs we will write in this course. C is not suitable for large object-oriented design projects that you might do in a system analysis or software engineering class. C is not an object-oriented language.

- A C program consists of global declarations (generally for constants and arrays), and a set of functions. One function must be called main(). I usually put main() at the end of my program, with any ancillary functions preceding it. Small programs can consist of just the main() function itself.

- After you have edited your C program, there are two things you need to do: compile and run. To compile a C program, use the command gcc, as in `gcc hello.c`. By default, the name of the executable file is `a.out`. To run a.out, type `./a.out`. (The ./ at the beginning of the command tells Linux that you want to run the a.out that is in the current directory.) If you want to specify a different name for your executable file, use gcc's –o option like this:
  `gcc –o hello hello.c`.

- Comments in C begin with /* and end with */. Newer C compilers also allow you to have line comments starting with //.

- Variables must be declared at the beginning of a function, not in the middle!

- I/O is usually performed using the built-in functions scanf() and printf().

- With scanf, it's generally good to just scan for one variable's value at a time, like this:
  ```
  scanf("%d", &int_variable);
  ```
  Note that you must put an '&' character before the name of the variable.
  %d is the code for an integer.

- The function printf() works just like System.out.printf() in Java. For example, this call
  ```
  printf("%4d %-12s\n", int_value, string_value);
  ```
  says to print an integer and then a string on the same line, and note that \n means that this line is finished. The 4 in %4d means to leave enough space for up to a 4 digit number, right justified. If your number has more than 4 digits, they will be printed anyway, but this may make your output appear out of alignment. The -12 in %-12s means to leave enough space for up to 12

characters in a string, left justified. This statement will also print a space between the integer and the string because we put a space between the %4d and %-12s.

- To perform I/O on a file, you need to use the `FILE*` type. Then you use the open() and close() functions. The syntax for open() is actually like that in Python. The first parameter is the name of the file, and the second parameter is either "r" or "w" to tell whether we are reading from or writing to the file. The nice thing about working with files is that doing I/O on files is just like standard I/O. The functions fscanf() and fprintf() are analogous to scanf() and printf(). They have an additional parameter at the beginning, which is the name of the appropriate FILE* variable. In other words, we have to specify which file we are doing I/O on.

- In your prior programming experience, you are used to reading input by scanning entire lines and then tokenizing them. The same can be done in C, but we may not often need to do so in this course. The `fgets()` function can read a whole line of text from standard input or from a file. And there is another built-in function `strtok()` that can tokenize. I can show you more about this later if you'd like.

- When designing a function, note that all parameters are technically passed by value. If you want your function to modify the value of a parameter (and have this value available to the calling function), then you must pass it as a pointer (i.e. memory address). In other words, using the "*" operator. For example, here is how we would implement a swap function for integers.

```
void swap (int * a, int * b)      // int* means pointer to int
{
  int temp = *a;    // *a here means the number pointed to by a
  *a = *b;
  *b = temp;
}
```

  One should note that in order to call the above function, you need to pass the *address* of variables. So, to swap the values of integers x and y, you would say `swap(&x, &y);` .

- There is no string type in C. Instead, we use an array of char. The last character needs to be the null character '\0' to signify the logical end of the string. You can allocate more space than you actually use. For example, for a line of text I customarily allocate 81 bytes. One for the null character, and up to 80 for whatever text might be on the line.

- Common include files are:

```
#include <stdio.h>              // for I/O
#include <string.h>
#include <stdlib.h>             // for random numbers and atoi
```

- The function atoi() is used to convert from a string to an integer, if that string has just an integer in it such as “16”. It is analogous to Integer.parseInt() in Java.

- There is no boolean type in C. Just use the int values 0 and 1. At the top of the program you can define your own constants, for example YES and NO if you like.

```
#define YES 1
#define NO  0
```

- As in Java, please beware of the two C operators = and ==. The = operator is used for assignment, and == is used for comparison. A common mistake is to use = when you mean to use ==. For example, `if (a = b)` is probably wrong, and the compiler won’t warn you.

- Although there are no classes, C has a rudimentary feature like a class. It’s called a struct. It only has attributes. You access an attribute with the dot operator, as you would in Java. Note that C has no concept of public/private/protected. Essentially everything in your source file is public. (C enforces information hiding by allowing you to compose your source code into multiple source files. But in this course, this concept will not be necessary.)

- Array syntax: for example, here is how to declare an array of 10 integers.

  ```
  int a[10];
  ```

  When declared globally, all cells are initialized to zero. Note that any variable declared inside a function is not initialized to anything in particular, so it’s “garbage.” Initialize as needed. The C compiler won’t warn you that a variable is not initialized.

- Common string functions:
  strlen(s) – returns the number of characters in s, up to but NOT including the null character
  strcpy(s1, s2) – sets the value of s1 equal to s2, so it works like an assignment statement. Please do it this way and never say s1 = s2, because this leads to an aliasing problem.
  strcmp(s1, s2) – compares strings, like a comparator object in Java. If s1 comes before [after] s2 in the alphabet, strcmp() returns some negative [positive] integer. The function returns zero if the two strings are identical.
  To access a single character, remember that a string is really just an array of char.

- Random number generating.
  At the beginning of main(), you should call srand(time(0)) to set the random number generator’s seed. Otherwise your program might run the same way every time. When you are ready to call up a random number, the rand() function automatically gives you a random nonnegative integer (which could be very large). To restrict the range to 0..n-1, you should do this: rand() % n. For example, if you want to roll a die to get a number 1-6, try this: rand() % 6 + 1.

- Command line arguments are similar to those in Java but with different syntax. If you want your program to use command line arguments, then you need to change the declaration of main to: main(int argc, char ** argv)
  The variable argc is the number of command line arguments, INCLUDING the name of the executable. The value of argc has to be at least 1. If you just run with "./a.out" then argc equals 1. If you run as "./a.out help 5", then argc equals 3. This is unlike Java which doesn't include the word java or the name of the main class file as command line arguments.
  The variable argv is essentially an array of strings (hence, a 2-d array of character). The value of argv[0] is the first command line argument, usually "a.out", etc.