Printing

coursera.org/learn/programming-fundamentals/supplement/XKeBN/printing

Our example programs so far have computed results, but had no way to communicate them to the user. Such programs would be useless in practice. Real programs have means to communicate with their user, both to read input and to provide output. Many programs that you are accustomed to have Graphical User Interfaces (GUIs), however, we will work primarily with programs that use a command line interface. Writing GUIs is a more complex task, and requires a variety of additional concepts.

Command line programs provide output to their user by printing it out on the terminal. In C, printing output is accomplished by calling the **printf** function, which takes a string specifying what to print. We will learn more about strings later, as they require knowledge of pointers to understand. For now, you can think of them as being text—words, or sentences. In much the same way that we can write down numerical literals (such as 3, or -42), we can write down string literals by placing the string we want inside of quotation marks, e.g., "This is a string". If we wanted to print out the string, “Hello World”, we would type **printf("Hello World");**.

The f in **printf** stands for “formatted,” meaning that **printf** does not just print literal strings, but can take multiple arguments (of various types), format the output as a string, and print the result. To format output in this way, the string argument of **printf** (which is called the “format string”) includes special *format specifiers*, which start with a percent sign (%). For now, we will only concern ourselves with %d which specifies that an integer should be formatted as a decimal (base 10) number. For example, if we wrote the following code fragment:

```
int x = 3;
int y = 4;
printf("x + y = %d", x + y);
```

it would print x + y = 7, because it would evaluate the expression x + y to get 3 + 4 which is 7, and format the number 7 as a decimal number in place of the %d specifier. The rest of the string is printed literally.

Another type of special information we can include in the string is *escape sequences*. Escape sequences are two (or more) characters, the first of which is a backslash (\), which gives the remaining characters special meaning. The most common escape sequence you will encounter is \n, which means “newline”. If you want your print statement to print a newline character (which makes the next output begin at the start of the next line), then you do so with \n. If you want a literal backslash (that is, you actually want to print a

backslash), \\ is the escape sequence for that purpose. We will note that you generally will want to print a newline in your output, not only so that it looks nice, but also because printf does not actually print the output to the screen until it encounters a newline character, or is otherwise forced to do so.

We will discuss the various format specifiers **printf** accepts, as well as the escape sequences that C understands in the next module.

The next video demonstrates the execution of code that prints output.