Arrays:

1. By capturing the length of the word, the length of the sentences, the position of punctuation etc, we can implement a program that takes a paragraph and determines what level of reading it is.
2. Cryptography – scrambling or encoding a message in a way that it can be sent safely and securely even though it may get intercepted.

Compiling:

1. When we use the make command to make an executable file, we are actually using the compiler called clang. When we run it, we get an output file named a.out, which means assembler output. If we run ./a.out, then the program itself will run.
2. Essentially, the make command runs the clang program, and then renames a.out to whatever your file name is.
3. Command line arguments are just words or prompts that you enter on the command line that modify the behavior of your program.
4. To do the assembling and renaming part of the program which make does, without using make, we need to write **clang –o hello hello.c**
5. An issue with the clang command is that if you are using third party libraries in your program, then clang will not recognize it. You will need to mention it explicitly.
6. To mention it explicitly, on the command line, we type **clang –o filename filename.c –lcs50**
7. The **–l** means library.
8. The # symbol means preprocessor directive.
9. To turn source code to binary, there are 4 steps:
   1. Preprocessing – When we write #include<headerfile>, the header file actually contains the prototypes for whatever functions we need. So when we run the clang command, what happens is those functions are found and directly copy pasted into your code. This is the preprocessing stage. Preprocessing converts those #include lines into the prototypes of the functions we need.
   2. Compiling – After the required functions are copy pasted into your code from the previous step, in this step, your code is converted into assembly language (As in, MV Left, ADD string, SUB string etc.).
   3. Assembling – In this step, the assembly code from above is converted into 0’s and 1’s, i.e, binary.
   4. Linking – When we include our header files like stdio.h, there is actually a file called stdio.c that exists somewhere. That is also converted into binary. The final step includes “linking” this file and the binary file of the program we wrote.
10. If you try to reverse binary code to original source code, which is technically possible, you will still lose a lot of the variable names and function names, to the point that just writing the program yourself is easier than performing the entire reverse engineering process.

Debugging:

1. Printf is a great tool for debugging.
2. We can also use a debugger. In the cs50 course, the debugger is already configured, and we just have to use the debug50 command.
3. The debugger needs the code to already be compiled. This means it cannot help with **Syntax errors.**
4. To run the debugger:
   1. make filename (to perform the compilation)
   2. debug50 filename (to perform the debugging)
5. When we declare a variable without initializing it’s value, it will most likely contain a garbage value. So in the line of code below, h will most likely have some garbage value in it:
   1. int h = get\_int(“Height: “);
6. The reason for this is when we declare a variable, the memory it is allotted most likely contains the remnants of some past memory in it.
7. The third method for debugging is rubber ducking.

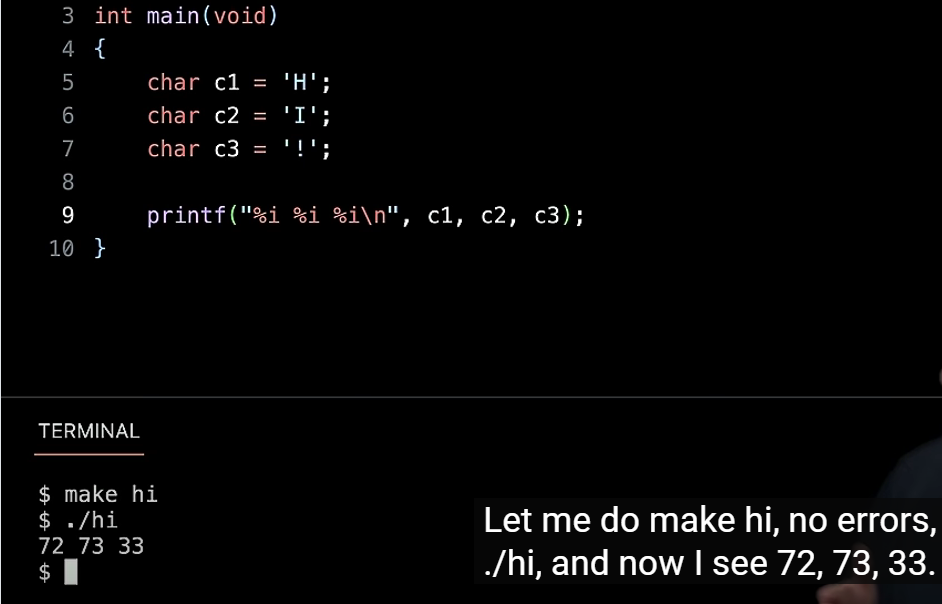
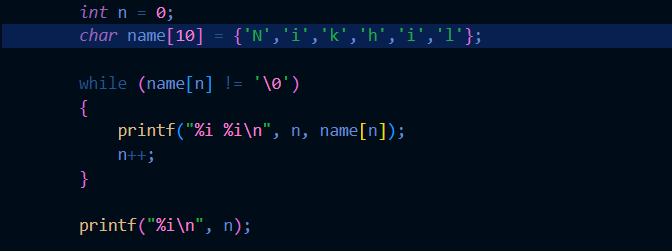
Memory:

1. Depending on what the programmer tells the software to store in a variable, and the context of the program, accordingly will the 0’s and 1’s be translated by the program.
2. bools take 1 byte. Although logically 1 bit is more than enough, it is easier for us to give it an entire byte. However, 7 of those bits are wasted.
3. If we do integer math, we will get an integer result, i.e, the digits after the decimal point will be discarded. This is why we should use typecasting.

Arrays:

1. An array is a sequence of similar data type values back to back in memory.
2. Syntax:-
   1. int arrayname[3];
   2. int arrayname[] = {1,2,3};
3. Oftentimes, it is better to store the array\_size as another variable in and of itself.
   1. int N = 4;
   2. int scores[N];
4. Since C does not have a function to find the array length, we need to store it explicitly.
5. When you are passing an array as an argument to a function, we don’t need to know the array length in advance, we can just put square brackets.
   1. Example: float average (int length, int scores[])
6. In the example above, scores is the array we are to pass.

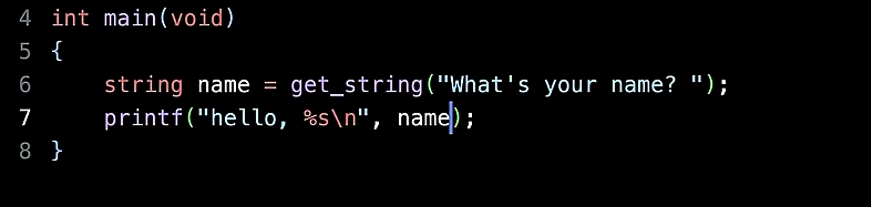
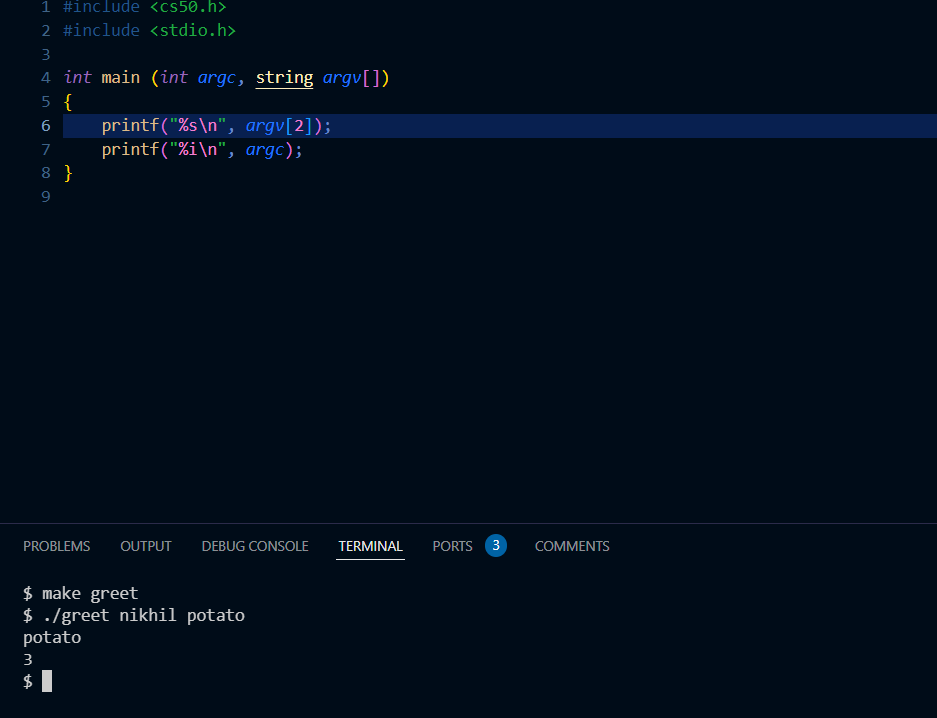
Strings:

1. A string is simply an array of variables of type char: an array of characters.
2. “%c” is to display a character, “%s” is to display a string.
3. 
4. In the example above, if we wrote “%c%c%c” in the printf statement, we’d get “HI!” as output. But because we used the integer formatting symbol “%i”, we get the ASCII values.
5. We can also do it the other way around, i.e, use %c for an int value provided.
6. Every string in C ends with \0, and this \0 is called NUL. We can even see this by printing it out, using printf and “%i”.
7. Also, every string is just a character array.
8. format specifies type 'int' but the argument has type 'char \*' – This error means that we are not using the correct format specifier. We need to use %s.
9. The syntax for a 2D array:
   1. char names[50][50] = {{'H','e','l','l','o'}, {'T','h','e','r','e'}};
10. 
11. Using the example above, we can find the length of a string.
12. We should however, use the library string.h. In it, there is a function called strlen()
13. 
14. We can initialize multiple variable inside the for loop. However, both have to be of the same type.

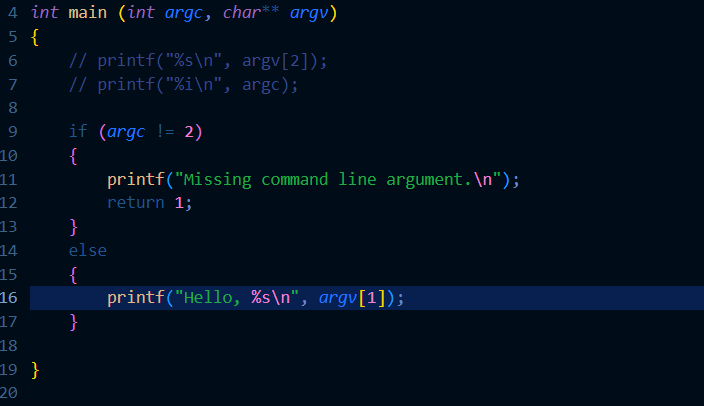
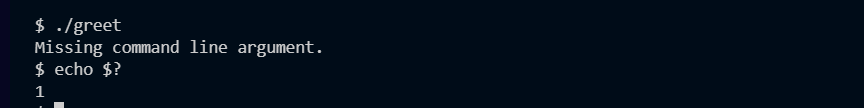
ctype.h:

1. toupper() = Makes the character you are on uppercase.

Command line arguments:

1. cd … … is an example of command line arguments, i.e, the cd command takes arguments.
2. rm filename is another example, the filename is the CLA
3. Another example is in int main(void)
   1. int main (int argc, string argv[]) //If we use cs50.h
   2. int main (int argc, char\*\* argv) //without cs50.h
4. The first argument is an int type, the second argument is an array of strings type.
5. argc means argument count – how many words did the user type at the prompt.
6. argv means argument vector. Essentially, it’s a list of command line arguments.
7. 
8. In the program above, we make the program, then execute it, and then we have to enter the user name.
9. 
10. In the above example, we use the cs50 library to include the string data type, and then we pass string argv[] as the second argument. When we look at the terminal, we see that we can directly pass Nikhil as an argument instead of prompting the user separately.
11. 
12. The example above is how to do it if we do not use the cs50.h library.
13. Note: We do not have to use the names argc and argv but it is general convention
14. 
15. If you notice in the example above, the arguments passed in the command line are the filename, Nikhil and potato. argc automatically realizes that there are 3 arguments.
16. cowsay –f duck quack
17. The above command can make your cow and duck 😐.
18. cowsay –f dragon roar
19. The above command is, you know what just go try it man damn.

Exit status:

1. These are the different exit codes that come at the end of program execution. 0 usually means the program executed fine, any other number indicates that some error occurred during execution.
2. 
3. The example above shows us how to potentially use exit statuses.
4. When we return any exit code, like 0 or 1, it secretly just exits the program and does not display. We can display it by using the command echo $?
5. 
6. The above is the output if enough arguments are not passed.
7. Status codes are another tool to help you debug your program.

Cryptography:

1. cipher – the algorithm responsible for converting plaintext to ciphertext. It encrypts it such that someone else can actually decrypt it.