When using fgets() to read a user input, newline is always added at the end of the string

- H E L L O \n is recorded from user input
    - The user typed H, E, L, L, O
    - C added \n at the end of the string
    - Zero terminator/null character
        - \0
        - How C marks the end of a string
    - To hold a string of n characters, you need n+1 slots in the array to account for the zero terminator
        - So “HELLO\n\0”’s size is 7

Truncating (Shortening) a String:

- Put a zero terminator anywhere you want
- `input[10] = “HELLO\n\0”;`
- `input[4] = ‘\0’;`
    - input becomes “HELL\0”
- Can use this to get rid of new line character
    - Place the \0 at (length of string) - 1
        - `input[6] = ‘\0’;`
        - Now the input string is “HELLO\0”

Get the length of a string:

- Java uses .length()
- In C, you have to count the characters until the zero terminator
    - `#include <string.h>` will let you use strlen()
        - Try not to use strlen() in a loop since it has linear runtime
    - But otherwise, do this:
        - `char input[100];`
        - `fgets(input, 100, stdin);`
        - `int len = strlen(input);`
        - `input[len - 1] = ‘\0’;`
        - Would you do this every time you needed to read a line from the console?
            - NO, put this in a function and call the function whenever you need to read a line from the console

Functions:

- A named piece of code with inputs (parameters) and outputs (return values)
    - Names show intent
- Useful problem solving tool
- Helps us to understand what the code is doing when looking at it

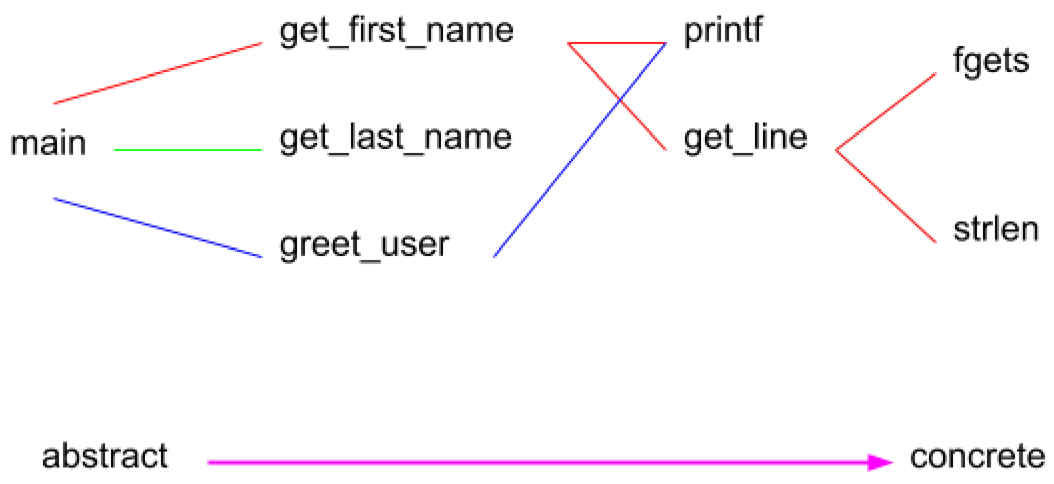
Abstraction

- Hiding details
- What to do vs. how to do it
- Focusing more on the WHAT and the WHY and less on the HOW

Call graphs:

- One way to structure a program is top-down

        - Start with the most abstract function, and then split it up over and over to the more concrete



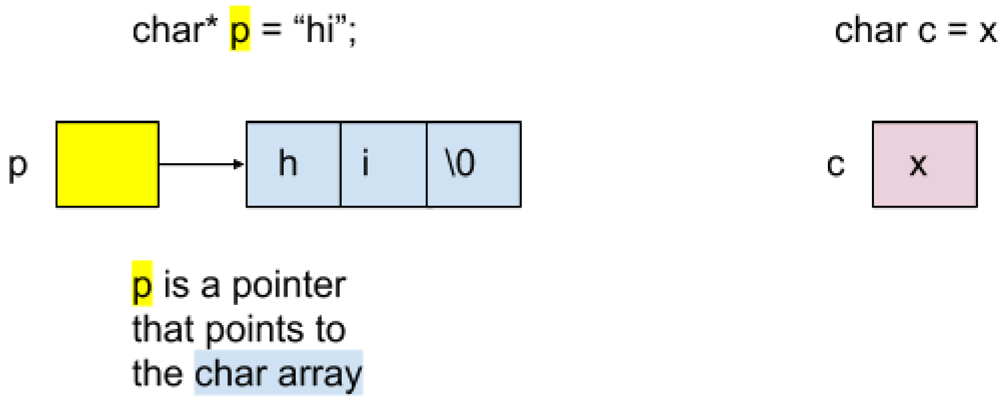
        -
        - It’s okay to call functions you haven’t written yet
            - Think of a function like a scene in a movie
                - Every line of code is like a new actor coming out and saying a line
                - Too many actors make it too crazy

Naming:

- Code is there for your benefit
- Make it easy to read
- Name your functions according to what they do and to what
- Avoid hard-coding important numbers, use constants
- Use camelCase or snake_case

Functions in C:

- Make a better fgets
    - Some issues:
        - How do you know how long to make the array?
        - Can you return an array in C? (No)
        - If you can’t return an array, how do you take one as an argument?
        - We want to do something like `get_line(input, 100);`
- C doesn’t treat arrays as objects like in Java
    - Instead, C uses pointers
        - Pointer: variable which holds a memory address; a reference to a thing
            - We can access data through the pointer
        - Arrays become pointers when they are passed to functions
        - And pointers are written like this
            - `void get_line(char* input, int size)`
                - The asterix
                - `char* input` is a pointer to the string input



        -
    - Improving fgets:
        - `void get_line(char* input, int size) {`
            - `fgets(input, size, stdin);`
            - `int len = strlen(input);`
            - `input[len - 1] = ‘\0’;`
            - `}`
            - input is a pointer to a char array
- Function prototypes:
    - C doesn’t know about functions if you try to access them before you declare them
        - It will still compile and run, however
        - If you try to access before you declare it, it assumes they have the signature `int name()`
    - If you want to access a function before declaring it:
        - `void get_line(char* input, int size);`
            - Put a semicolon at the end, then write functions which use get_line
        - But, it’s usually a better idea to reorder your functions

Returning Arrays?:

- Can’t do it like in Java
- Can’t do with a local array
    - Will make more sense when discussing stacks later on
    - Local variable disappears, now have a pointer to something that doesn’t exist
        - Accessing a null pointer has undefined behavior
- Returning an array is so bad in C that gcc will force your function to return null if you try to do it directly
- Don’t return arrays