How do our programs communicate with the outside world? (beyond its instructions / CPU)

* UNIX - Sends things out through byte-streams, usually in streams of 3
  + 2 output streams
  + 1 input stream
* These streams in standard C library
  + Std Input, Std Output, Std Error

Write (C)

* Write is a direct request to the operating system
  + “Send some bytes out on a stream”
* Example: *write (1, “Andrew Rocks\n”, 13);*
  + Sending out 13 bytes
  + Bytes are 8 bit numbers, unsigned 0 – 255.
* What values am I requesting on that stream?
  + “ “ (double quote) is shorthand to give me an array of bytes containing ASCII CODES for characters.
* The example means: Create a 13 element array, initialised to the ASCI CODE “A” “n” “d” etc.
  + You cannot assign to an array created in this way, you will seg fault:  
    *char \*p = “Andrew Rocks\n”****p[5] = ‘Z’;***<-- Line with error *write(1, p, 13);*
  + It is a **Runtime Error**, which is an error that occurs during the execution of a program.
  + This is because the “write” command puts the data in an area of memory that is marked “read only” and not to be changed. An attempt to change it will produce an error from the operating system.
* Why not use “Write” in my C program?
  + “Write” may not be available on other platforms – It is UNIX specific
  + On a standard C program, we would use “Printf” instead of “Write”
* Does every C implementation have a “Printf”?
  + No.
  + In an embedded system (e.g. a Microwave) that runs on C, it may not have “Printf”.
  + The embedded processor may be very tiny and can only store very limited information, therefore it excludes code for “Printf”.
  + How do I know “Printf” goes to Standard Output stream?
    - It is part of the function in Stdio.h.
  + You may want to change “Printf” to go to another stream:
    - E.g. printf(stderr, “Andrew Rocks\n”);

Every program starts off with Stdio, Stderr setup somewhere behind the scenes. How do they get created?

* They get created in the **C Runtime Environment**
* When GCC creates a C program, before your code in Main Function is called, it includes some code it runs itself, which is effectively setting up the “C Runtime Environment” / sets up Stdio Stderr and some code that you don’t control.
* Where else would you like streams going with your program?
  + Might want to send stuff to devices
  + Sending stuff to another computer – over the network
  + You can send a stream of bytes to anywhere else in the world.
* These byte streams are one form of communication for your program. There are a couple of other avenues of communication for your running program in UNIX.
  + A small integer “0” is normally used to indicate success or error.
  + “0” usually indicates OKAY and a non-zero value indicates some sort of error.
  + You can also use “exit(0)” to terminate a program.

Shell Syntax

* “ -l “ (dash L) argument that specifies Long. It changes the behaviour of a command.
  + E.g. “ls -l “ would show a list of files with extended information.
  + Result would look something like:



* + What is a file?  
    🡪 A file is an array of bytes / zero or more bytes in a sequence.
  + In the above “ls –l” file information example:

🡪 The first file is an array of 154097 elements, each of which are bytes.

🡪 It also shows a date, which shows when the file was last modified.  
🡪 It also shows permissions, i.e. who can read / write the file.

* **“ < “ (less than)**, which means take your input from that file, run this program and standard input will no longer come from the terminal, but come from that file.
* **“ > “ (greater than)**, redirects the output to a file.
* **“ >>” (append to the file)**
* **“ | “ (pipeline)** for example:
  + “date | wc”. Running two programs simultaneously
  + It will run those two programs, such that the standard output of the first program, is a string of bytes which go into the standard input of the second program.
* Pressing “Ctrl D” causes the operating system to simulate end of input as though you’re reading from a file.
* **“cat” (Concatenate)** If you had 10 files that you want to turn into 1 file, you can do this:

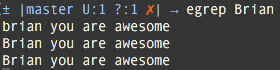


* **“echo”** reads its arguments and prints them as standard output.

**Look through notes and incorporate other information in / things that are left out**

Egrep

* Egrep is a command that prints lines that matches a pattern.
  + E.g. matching the string pattern “Brian”



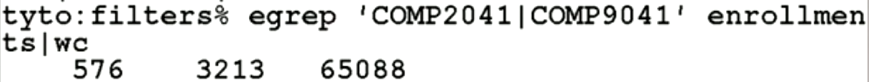
* + First string doesn’t match, due to case sensitivity
  + Second string matches, therefore it is printed
* Egrep’s first argument is always a pattern, but any subsequent arguments are interpreted as filenames.
  + E.g. matching the string pattern “main” in the C file “testFile.c”



* “wc” is a command that counts the number of lines in the file that match the specified pattern.
  + E.g. counting the number of COMP2041 students in a student database



* + 477 = no. of lines with the pattern
  + 2648 = no. of words with the pattern
  + 53901 = no. of characters with the pattern.
  + From this command, we can tell that there are 477 enrolled in COMP2041.
* How do I match for both COMP2041 COMP9041?



* The pipeline above is used as an alt, e.g. COMP2041 OR COMP9041
* “ \* “ (Asterisk) means repetition. Zero or more repetitions of the PRECEEDING pattern.
  + E.g. ‘ba\*d’



* + This pattern matches bd, bad, baad, baaaaaad and more repetitions of ‘a’ before ‘d’
  + It is an infinite set.
* Round brackets in regular expressions have a special meaning, e.g**. ‘b(ab)\*d’**
  + Used for grouping ??? More on this point?

Regular Expressions

* A regular expression is a way of specifying a set of strings. It may be a small or large set of strings.

**Make additional study notes for:**

* **TAIL**
* **HEAD**
* **CAT**
* **WC**
* **CUT**
  + **The cut utility cuts out selected portions of each line (as specified by “list”) from each file and writes them to the standard output.**
  + **-f list = list specifies fields, separated in the input by the field delimiter character. Output fields are separated by a single occurance of the field delimiter character.**
* **GREP / EGREP**
  + **Start with = “^text”**
  + **End with = “text$”**