Command Line Arguments

Command Line Arguments

- Arguments provided to your program via the command line.
  - Not read from user input
- in C/C++:
  - **`argv[]`** contains program name and arguments
  - **`argc`** contains number of arguments plus one
- in Perl:
  - **`@ARGV`** contains list of arguments
  - **`$0`** contains program name

Examples:

- **`myscript.pl 15 4 hello`**
  - **`@ARGV`** ➔ (15, 4, 'hello')
  - **`$0`** ➔ 'myscript.pl'
  - **`scalar(@ARGV)`** ➔ 3
- **`perl_hw2.pl`**
  - **`@ARGV`** ➔ ( )
  - **`$0`** ➔ 'perl_hw2.pl'
  - **`scalar(@ARGV)`** ➔ 0

Notes

- These are true globals – no matter what package you're in, **$0** always means **$main::0**, **@ARGV** always means **@main::ARGV**
  - unless you've declared a lexical **@ARGV**
    - don't do that.
- Depending on your system, **$0** may contain just the script name, a relative path to the script, or an absolute path.

Some Magic

- The standard **<>** operator has some magic built into it
- When "empty", **<>** will open the first file specified on the command line, and begin reading it.
  - Once it's exhausted, will open the next file, etc.
  - If another call to **<>** occurs after all arguments are read, **<>** begins reading from **STDIN**
- While **<>** is processing command line arguments, **$ARGV** contains the name of the current argument.
- If any file can't be opened, a warning is issued, and processing continues with the next one.

Magic Example

- **myscript.pl file1.txt sample**
- ```
  while (<>){
    chomp;
    print "$ARGV, line $. = $_\n";
  }
  ```
- open file1.txt, print out all lines in that file
- open sample, print out all lines in that file
- loop terminates.
- At this point, **@ARGV ➔ ()**
  - any future reads to <> will read from STDIN
  ```