Lecture 1: Introduction to Basic C Programs

Review and Preview

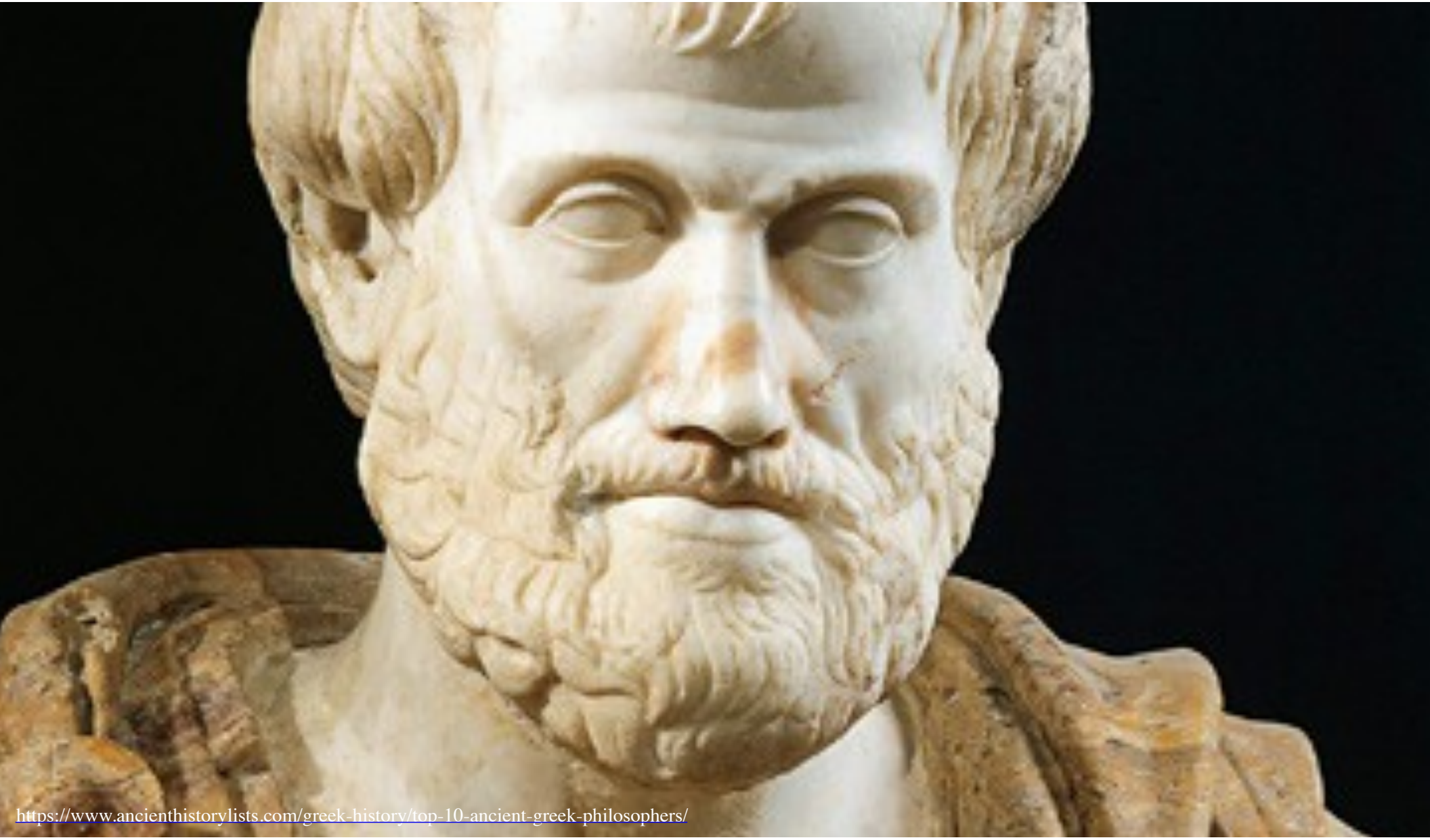
- #include, #define
- libraries, header files
- compiling, execution
- directory structure
- pipes
- arguments
- arrays
- for loops

K&R: Chapter 1
Sections 1.1 to 1.5

UNIX Command of the Day: `more`

- `more` allows you to view a file one page at a time.
- There is another related program called `less` than functions like `more` but gives you more options. Truly less is more.
- With both commands you page through a file by hitting the space bar.

```
$ more filename.txt
```



https://www.ancienthistorylists.com/greek-history/top-10-ancient-greek-philosophers/

UNIX Philosophy

Why use UNIX to teach programming?

The Basics of the UNIX Philosophy

- Write programs that do **one** thing and do it **well**.
- Write programs to work **together**.
- Write programs to handle **text** streams, because that is a universal interface.

Doug McIlroy, the inventor of UNIX pipes and Ken Thompson, co-developer of UNIX

The Philosophy according to Brian Kernighan

1. Everything is a **file**.

2. Small, single-purpose programs (**modularity**).

3. Ability to chain programs together to perform complex tasks (**piping**).

4. Avoid captive user interfaces - most UNIX programs are non-interactive (requiring arguments instead of user input) which makes them useful within **scripts**.

https://www.brandt.us/commercial-plumbing-contractors/hygienic-pipes-fittings/

Unix Pipes

How to take advantage of STDIN and STDOUT

UNIX Pipes

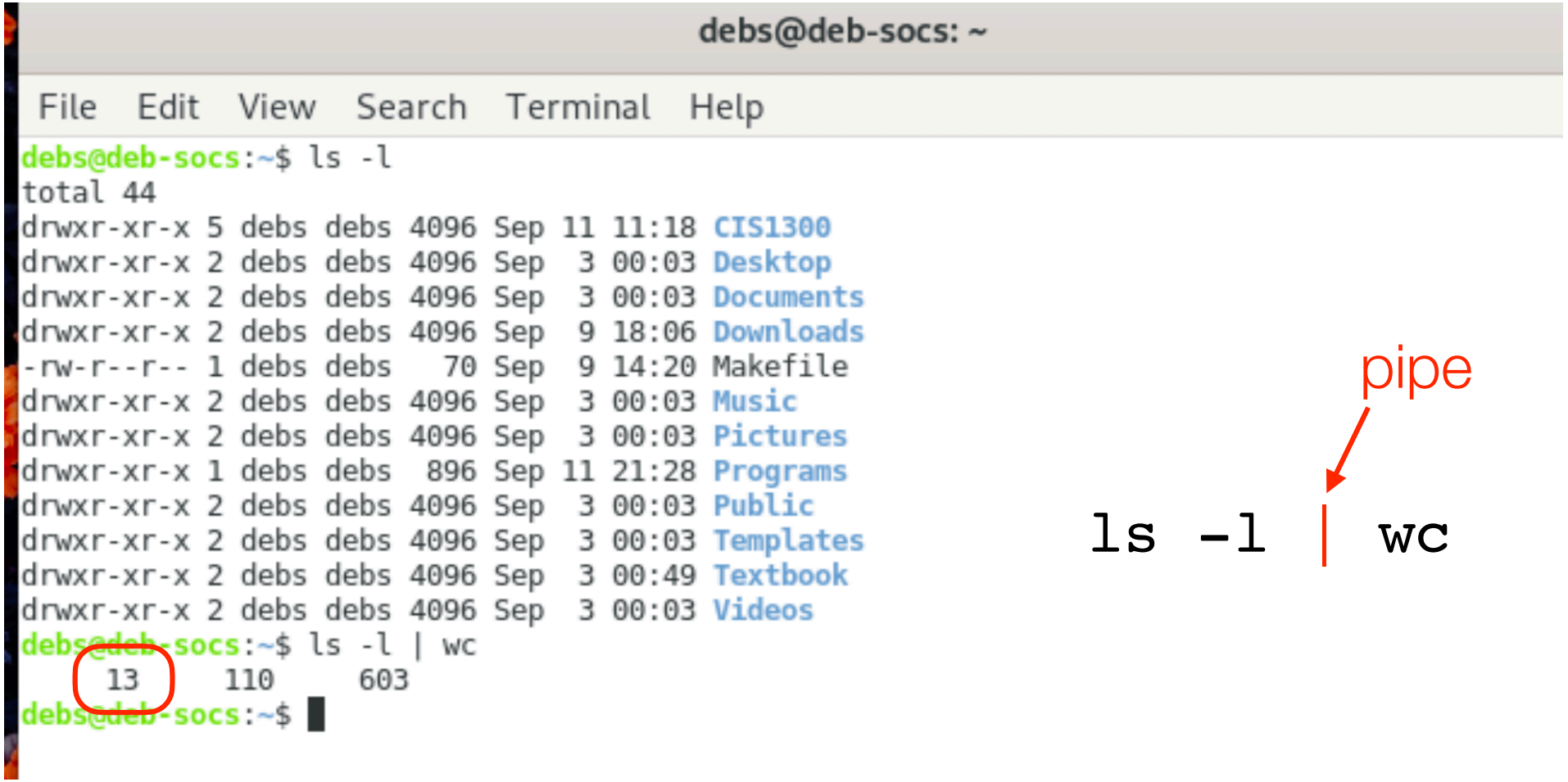
- A pipe redirects output (**STDOUT**) from one program to the input (**STDIN**) of another program.
- This direct connection between commands/programs/processes allows for **simultaneously** operation.
- Data is transferred **continuously**. The other approach would be to transmit the data from one program to another through **temporary** text files.

Unix Pipes

- The direction is one way (from left to right) and the sequence of programs is referred to as a pipeline.

  ```
  ls -l | wc
  ```

  - `ls -l` will list the files in the current directory using the long format
  - `wc` will print out the number of lines, words and characters generated by `ls -l`



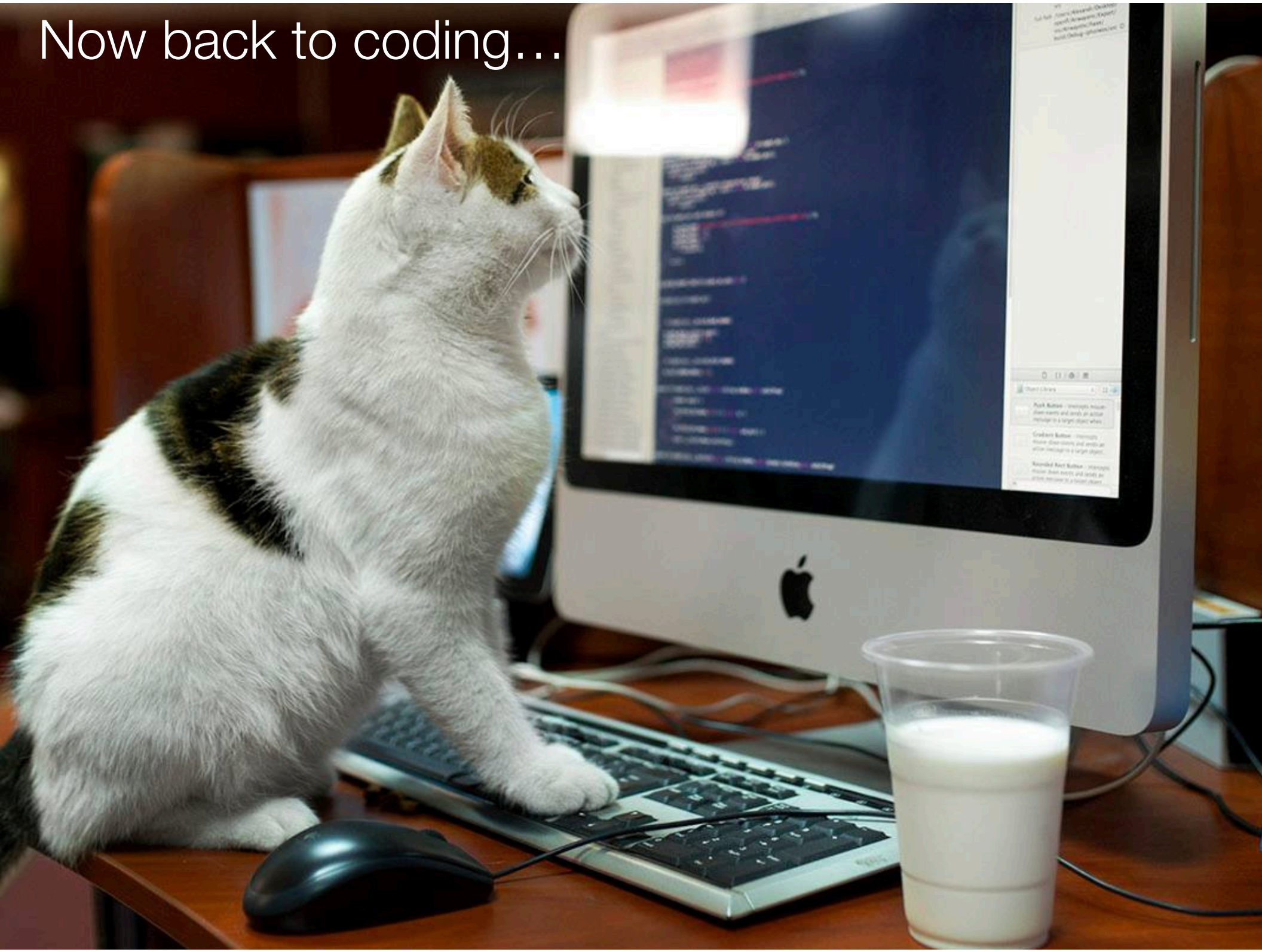
pipe

`ls -l | wc`

`ls -l` : list files in long format

`wc` : prints out number of lines, words, characters

Now back to coding…



```
main(int argc, char *argv[])
```

What does it really do?

Let's find out…

```
debs@deb-socs:~/CIS1300/FirstPrograms$ cat args.c
#include <stdio.h>

int main ( int argc, char *argv[] ) {

        printf ( "Number of command line arguments = %d\n", argc );

        printf ( "argv[0] = %s\n", argv[0] );

        return ( 0 );
}
debs@deb-socs:~/CIS1300/FirstPrograms$ gcc -ansi -o args args.c
debs@deb-socs:~/CIS1300/FirstPrograms$
debs@deb-socs:~/CIS1300/FirstPrograms$ ./args 1 2 3 4 5
Number of command line arguments = 6
argv[0] = ./args
debs@deb-socs:~/CIS1300/FirstPrograms$
```

Number of arguments

First argument

First argument is the name of the program

But really, what is `char *argv[]` ?

- Variables contain various types of information.
- `int` stores an integer, e.g. `int flag;`

```
flag = 1;
```

- `char` stores a character, e.g. `char letter;`

```
letter = 'a';
```

- `char *argv[]` stores an array of character strings *(well not really, it stores an array of pointers to arrays of chars)*

What is an Array?

- If a variable stores one instance of a particular type of thing (such as an integer), an **array** stores an **ordered collection** of things.

- **Collection** means that there is more than one and **ordered** means that they are in a specific order and can be referenced by an **index**.

- `int grades[5];` `grades[0] = 100;`

- `int grades[5] = { 100, 90, 45, 75, 92 };`

```
debs@deb-socs:~/CIS1300/FirstPrograms$ cat arrays.c
#include <stdio.h>

int main ( int argc, char *argv[] ) {

   int grades[5] = { 100, 90, 45, 75, 92 };
   int i = 0;

   for ( i=0; i<5; i++ ) {
      printf ( "grades[%d] = %d\n", i, grades[i] );
   }

   return ( 0 );

}
debs@deb-socs:~/CIS1300/FirstPrograms$ gcc -ansi -o arrays arrays.c
debs@deb-socs:~/CIS1300/FirstPrograms$
debs@deb-socs:~/CIS1300/FirstPrograms$ ./arrays
grades[0] = 100
grades[1] = 90
grades[2] = 45
grades[3] = 75
grades[4] = 92
debs@deb-socs:~/CIS1300/FirstPrograms$
```

Control: `if…else`

- The most basic program control structure is `if…else`.
- Basic structure:

```
if ( boolean expression ) {
    Statements to execute if boolean is true
} else {
    Statements to execute if boolean is false
}
```

Control: `if…else` boolean expressions

- What is a boolean expression?
  - These expressions evaluate to either **true** or **false**.
    - Example: `a > 10`
  - There are also boolean operators that can be used with multiple boolean expressions [3].

Control: `if…else` boolean operators

- **OR** - true when either boolean expression is **true**

```
if ( a > 10 || b > 10 ) …
```

- **AND** - true when all boolean expressions are **true**

```
if ( a > 10 && b > 10 ) …
```

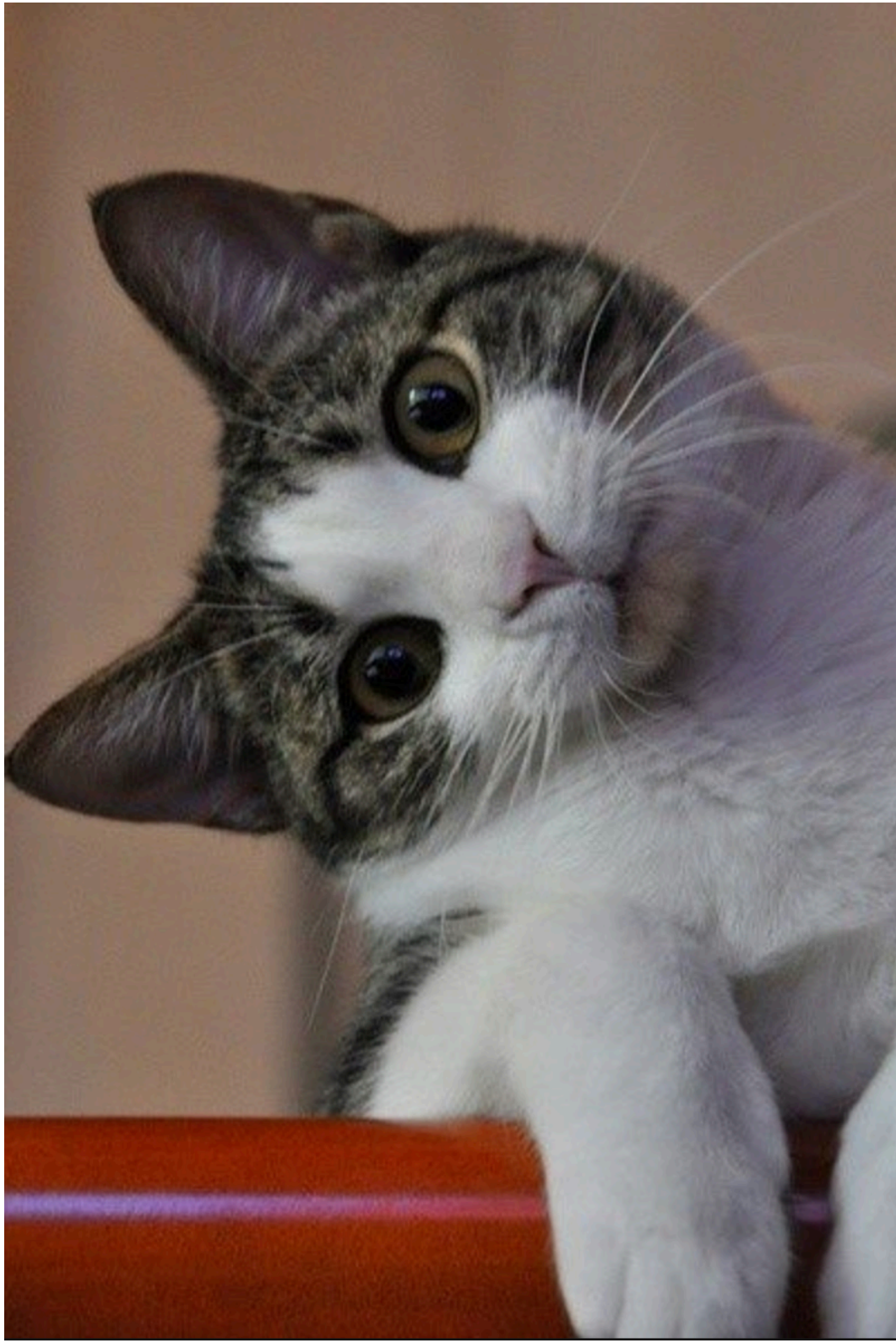
- **NOT** - true when boolean expression is false

```
if ( !( a > 10 && b > 10 ) ) …
```

Control: `if…else` Chains

```
if ( boolean expression 1 ) {
   Statements to execute if expression 1 is true
} else if ( boolean expression 2 ) {
   Statements to execute if expression 2 is true
} else {
   Statements to execute if none are true
}
```

Back to command line arguments…



```
#include <stdio.h>
int main ( int argc, char *argv[] ) {

   int i = 0;

   printf (“The number of args = %d\n”, argc);

   for ( i=1; i<argc; i++ ) {
      printf (“argv[%d] = %s\n”, i, argv[i]);
   }

   return (0);
}
```

```
#include <stdio.h>
int main ( int argc, char *argv[] ) {
    int i = 0;
    printf (“The number of args = %d\n”, argc);
    for ( i=1; i<argc; i++ ) {
       printf (“argv[%d] = %s\n”, i, argv[i]);
    }
    return (0);
}
```

```
$ ./args cat dog 2 horse
```

The number of args = 5
argv[1] = cat
argv[2] = dog
argv[3] = 2
argv[4] = horse

```
#include <stdio.h>
#include <stdlib.h>

int main ( int argc, char *argv[] ) {
   int i = 0;
   int j = 0;
   printf ("Number of arguments = %d\n", argc);

   for ( i=1; i<argc; i++ ) {
      j = atoi(argv[i]);
      printf ("argc[%d] = %d\n", i, j);
   }
   return ( 0 );
}
```

atoi( ) - ascii to integer - converts a string to an integer

```
#include <stdio.h>
#include <stdlib.h>

int main ( int argc, char *argv[] ) {

   int i = 0;
   int firstNum = 0;
   int nextNum = 0;
   int product = 0;

   firstNum = atoi ( argv[1] );
   nextNum = atoi ( argv[2] );
   product = firstNum * nextNum;
   printf ("%d X %d = %d\n", firstNum, nextNum,
product);

   return ( 0 );
}
```

```
#include <stdio.h>
#include <stdlib.h>

int main ( int argc, char *argv[] ) {

   char *monthName[12] = { "January", "February", "March", "April", "May",
                "June", "July", "August", "September", "October", "November",
                "December" };

   int monthLength[12] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };

   int dd = 0;
   int mm = 0;
   int yyyy = 0;
```

```
    if ( argc < 4 ) {
        printf ( "Usage: ./dates mm dd yyyy \n" );
        return ( 1 );
    } else {
        dd = atoi ( argv[1] );
        mm = atoi ( argv[2] );
        yyyy = atoi ( argv[3] );
    }

    printf ( "The date is %02d-%02d-%04d\n", dd, mm, yyyy );

    printf ( "In %s there are %d days\n", monthName[mm-1],
                                          monthLength[mm-1] );

    return ( 0 );

}
```

The First Marks of the Semester Approach...

Lab 2 - on CourseLink now!

Review and Hotwash

What we covered today and what you thought was important.

https://www.amazon.com/Review-My-Cat-Tanner-Ringerud/dp/1402285361