2510 · C Lecture 5 ||

• Command Line Arguments

```
$ gcc -ansi -W -Wall -pedantic lab1.c
```

bash prompt

command-line arguments

↳ they are passed to the C program through main

2 VERSION OF MAIN

1. `int main(void) {...}`
2. `int main(int argc, char * argv[] {...}`

think of it like an array of strings for now

`argc` = argument COUNT (how many)
`argv` = argument VECTOR (how big)

- the cmd args are passed into **main** via `argv` and `argc` is the number of args

in the above example...

`argc` = 6 (b/c 6 cmd args)
`argv[0]` = "gcc"
`argv[1]` = "-ansi"
`argv[2]` = "-W"
...

STANDARD IDIOM TO PROCESS CMD LINE ARGS

```
int main (int argc, char * argv[]) {
    int i;
    for (i = 0; i < argc; i++) {        // Lab 1 → 0
      /* process argv[i] */
}
```

Example: Program to echo cmd-line args

$ echo hello world → hello world

Going through the cmd args to print everything

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

in cmd: $ ./myecho hello world

hello world \n

how to eliminate

To eliminate the extra trailing space, has to do w/ something in the loop:

```
for (i=1; i<argc; i++) {
    if (i == argc - 1) {
        printf ("%s\n", argv[i]);
    } else
        printf ("%s", argv[i]);
```

STANDARD IDIOM TO PROCESS CMD LINE ARGS

```
int main (int argc, char * argv[]){
    int i;
    for (i = 0; i < argc; i++){      // Lab 1 → 1
      /* process argv[i] */
}
```

Example: Program to echo cmd-line args

$echo hello world → hello world

Going through the cmd args to print everything

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

in cmd: $ ./myecho hello world

↓

hello world \n

how to eliminate

OR WE CAN USE THE TERNARY OPERATOR (? :)

BCIT COMPUTING bcit.ca/computing

```
for(i=1; i<argc; i++) {
   printf(i==argc-1 ? "%s\n" : "%s ", argv[i])
      or...
for (i=1; i<argc; i++){
 printf("%s %c", argv[i], i==argc-1?
                          '\n' : ' ');
```

... Lecture 5 ||

INPUT/OUTPUT

↳ performed via streams

- When a C program starts, 3 streams are made available:

1. **stdout** - associated with console output by default
2. **stderr** - " "
3. **stdin** - associated w/ keyboard input by default

↳ We can change these default associations using I/O redirection.
(This is good to separate output and errors into two files)

```
$ ./a <input >output 2>error
```

- `$` : bash prompt
- `./a` : our program
- `<input` : used to redirect stdin
- `>output` : used to redirect stdout
- `2>error` : 2> used to redirect stderr

* input, output, error are names of files
** you can choose any of 3 to redirect

- I/O redirection is not specific to C, its a feature of the shell
- convention is reg messages printed to **stdout**
  error messages printed to **stderr**

```
$ gcc -ansi -W -Wall -pedantic lab1.c 2>errors
```