Command-line Arguments

- Idea: get input from the command line before the program launches, rather than as the program executes
- These arguments are passed as arguments to `main()`
- May have seen `main()` automatically written as the following by your IDE:

```
int main(int argc, char *argv[]){
  // body
}
```

- `argc` refers to the number of command line arguments passed to your program (argument count)
  - `argc` is always at least 1, because the first argument in `argv` is the name of your program (which is always supplied)
- `argv` refers to the actual command line arguments (argument vector)
  - `argv[0]` is the name of the program (set when you compile - your IDE is usually taking care of supplying this name)
  - `argv[1]` is the first command line argument
  - `argv[2]` is the second command line argument
  - …
  - `argv[argc]` is the a NULL pointer
  - You can think of this as a tokenized version of the command line string by using `strtok` to deliminate on spaces

Basic example

```
#include <stdio.h>

int main(int argc, char *argv[]){
  printf("Program name is %s\n", argv[0]);
  // loop over remaining arguments, printing each
  if (argc > 1){
    for (int i = 1; i < argc; i++){
      printf("argv[%d]: %s\n", i, argv[i]);
    }
  }
}
```

- That's it!
- But we can have more sophisticated command line arguments. For instance, what if we wanted to allow a set of options that were specified in no particular order?
  - We could write this code ourselves, but we'd have to check for every possible argument at every `argv` element, and we may not handle edge cases, like repeated/conflicting options
- Fortunately, there is an existing library to help you do this

`getopt_long`

- C Library included with `getopt.h`
- We'll supply a list of possible arguments, whether or not this option has a required argument,

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

int main (int argc, char **argv)
{
  int verbose_flag;
  int c;

  struct option long_options[] =
  {
    /* These options set a flag. */
    {"verbose", no_argument,       &verbose_flag, 1},
    {"brief",   no_argument,       &verbose_flag, 0},
    /* These options don't set a flag.
       We distinguish them by their indices. */
    {"add",     no_argument,       NULL, 'a'},
    {"append",  no_argument,       NULL, 'b'},
    {"delete",  required_argument, NULL, 'd'},
    {"create",  required_argument, NULL, 'c'},
    {"file",    required_argument, NULL, 'f'},
    {0, 0, 0, 0}
  };

  while (1)
  {
    /* getopt_long stores the option index here. */
    int option_index = 0;

```

```
    c = getopt_long (argc, argv, "abc:d:f:",
                     long_options, &option_index);

    /* Detect the end of the options. */
    if (c == -1)
      break;

    switch (c)
    {
      case 0:
        /* If this option set a flag, do nothing else now. */
        if (long_options[option_index].flag != 0)
          break;
        printf("option %s", long_options[option_index].name);
        if (optarg)
          printf(" with arg %s", optarg);
        printf("\n");
        break;

      case 'a':
        puts("option -a\n");
        break;

      case 'b':
        puts("option -b\n");
        break;

      case 'c':
        printf("option -c with value '%s'\n", optarg);
        break;

      case 'd':
        printf("option -d with value '%s'\n", optarg);
        break;

      case 'f':
        printf("option -f with value '%s'\n", optarg);
        break;

      default:
        abort();
    }
  }
```

```
73
74    /* Instead of reporting '--verbose'
75       and '--brief' as they are encountered,
76       we report the final status resulting from them. */
77    if (verbose_flag){
78      puts("verbose flag is set");
79    }
80
81    /* Print any remaining command line arguments (not options). */
82    if (optind < argc)
83      {
84        printf("non-option ARGV-elements: ");
85        while (optind < argc){
86          printf ("%s ", argv[optind++]);
87        }
88        putchar('\n');
89      }
90
91    exit (0);
92  }
```