

CS50 Command-Line Interaction

Overview

When running a program from the command line, you've generally executed a command like `./program_name` at the command line. C also allows you to specify a program's **command-line arguments**, which allows the person running the program to pass arguments into the `main` function of the program by specifying the arguments at the command line. This offers an alternative means of providing input to a program beyond just requesting input while a program is running, such as with `get_string()`.

Key Terms

- command-line arguments
- argument count
- argument vector

`./hello`

argc	argv
1	0 ./hello

`mkdir src`

argc	argv	
2	0 mkdir	1 src

`clang -o hello hello.c`

argc	argv			
4	0 clang	1 -o	2 hello	3 hello.c

argc, argv

Many of the command-line programs that you have likely called before (`make`, `cd`, `clang`, `mkdir`) all take command-line arguments. In C, command-line arguments are passed into the `main` function as inputs. However, we've previously written our `main` functions to take no arguments (`void`).

To accept command-line arguments, we can revise the `main` function to take two arguments: `argc`, an integer, and `argv`, an array of strings.

`argc`, which stands for "**argument count**", represents the number of arguments passed into through the command line. Each word (separated by spaces) counts as its own argument, and the calling of the program itself (e.g. `./hello`) counts as an argument.

`argv`, which stands for "**argument vector**", is the actual array representing the arguments themselves. Each value in the array is a string.

If you were to look at `argc` and `argv` when calling a program with no arguments, like calling `./hello`, `argc` would be 1 (because the calling of the program is the only argument). `argv`, on the other hand, would be an array consisting of just one element: the string `./hello` stored at index 0.

If you were to look at `argc` and `argv` when calling a program that does have arguments, like calling `mkdir src`, `argc` would be 2, since two arguments are passed in via the command line, and `argv` would be an array with two elements: the string `mkdir` stored at index 0, and the string `src` stored at index 1.

Using Command Line Arguments

Shown to the right is an example of a program which accepts command-line arguments. Notice on line 4 that the definition of the `main` function has changed to include the arguments `argc` and `argv`. No size of `argv` is specified on line 4, so that any array, regardless of its size, can be passed into the `main` function.

Inside of `main` function, the program loops through the array, starting at index 0, and incrementing so long as `i < argc`. It's important to stop there, because the largest index of `argv` that you can access is `argc - 1` (since arrays are zero-indexed). During each iteration, the program prints out the value of `argv` at index `i`.

The result of the program is that each of the program's command line arguments is printed on a new line.

```
1 | #include <cs50.h>
2 | #include <stdio.h>
3 |
4 | int main(int argc, string argv[])
5 | {
6 |     for (int i = 0; i < argc; i++)
7 |     {
8 |         printf("%s\n", argv[i]);
9 |     }
10 | }
```