# Chapter 19: Command-line arguments

| Parameter | Details |
|---|---|
| argc | argument count - initialized to the number of space-separated arguments given to the program from the command-line as well as the program name itself. |
| argv | argument vector - initialized to an array of `char`-pointers (strings) containing the arguments (and the program name) that was given on the command-line. |

## Section 19.1: Print the arguments to a program and convert to integer values

The following code will print the arguments to the program, and the code will attempt to convert each argument into a number (to a `long`):

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

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

    for (int i = 1; i < argc; i++) {
        printf("Argument %d is: %s\n", i, argv[i]);

        errno = 0;
        char *p;
        long argument_numValue = strtol(argv[i], &p, 10);

        if (p == argv[i]) {
            fprintf(stderr, "Argument %d is not a number.\n", i);
        }
        else if ((argument_numValue == LONG_MIN || argument_numValue == LONG_MAX) && errno ==
ERANGE) {
            fprintf(stderr, "Argument %d is out of range.\n", i);
        }
        else {
            printf("Argument %d is a number, and the value is: %ld\n",
                   i, argument_numValue);
        }
    }
    return 0;
}
```

References:

- strtol() returns an incorrect value
- Correct usage of strtol

## Section 19.2: Printing the command line arguments

After receiving the arguments, you can print them as follows:

```
int main(int argc, char **argv)
{
    for (int i = 1; i < argc; i++)
    {
        printf("Argument %d: [%s]\n", i, argv[i]);
```

```
    }
}
```

**Notes**

1. The `argv` parameter can be also defined as `char *argv[]`.
2. `argv[0]` *may* contain the program name itself (depending on how the program was executed). The first "real" command line argument is at `argv[1]`, and this is the reason why the loop variable `i` is initialized to 1.
3. In the print statement, you can use `*(argv + i)` instead of `argv[i]` - it evaluates to the same thing, but is more verbose.
4. The square brackets around the argument value help identify the start and end. This can be invaluable if there are trailing blanks, newlines, carriage returns, or other oddball characters in the argument. Some variant on this program is a useful tool for debugging shell scripts where you need to understand what the argument list actually contains (although there are simple shell alternatives that are almost equivalent).

# Section 19.3: Using GNU getopt tools

Command-line options for applications are not treated any differently from command-line arguments by the C language. They are just arguments which, in a Linux or Unix environment, traditionally begin with a dash (-).

With glibc in a Linux or Unix environment you can use the getopt tools to easily define, validate, and parse command-line options from the rest of your arguments.

These tools expect your options to be formatted according to the GNU Coding Standards, which is an extension of what POSIX specifies for the format of command-line options.

The example below demonstrates handling command-line options with the GNU getopt tools.

```
#include <stdio.h>
#include <getopt.h>
#include <string.h>

/* print a description of all supported options */
void usage (FILE *fp, const char *path)
{
    /* take only the last portion of the path */
    const char *basename = strrchr(path, '/');
    basename = basename ? basename + 1 : path;

    fprintf (fp, "usage: %s [OPTION]\n", basename);
    fprintf (fp, "  -h, --help\t\t"
                 "Print this help and exit.\n");
    fprintf (fp, "  -f, --file[=FILENAME]\t"
                 "Write all output to a file (defaults to out.txt).\n");
    fprintf (fp, "  -m, --msg=STRING\t"
                 "Output a particular message rather than 'Hello world'.\n");
}

/* parse command-line options and print message */
int main(int argc, char *argv[])
{
    /* for code brevity this example just uses fixed buffer sizes for strings */
    char filename[256] = { 0 };
    char message[256] = "Hello world";
    FILE *fp;
    int help_flag = 0;
    int opt;
```

```c
    /* table of all supported options in their long form.
     * fields: name, has_arg, flag, val
     * `has_arg` specifies whether the associated long-form option can (or, in
     * some cases, must) have an argument. the valid values for `has_arg` are
     * `no_argument`, `optional_argument`, and `required_argument`.
     * if `flag` points to a variable, then the variable will be given a value
     * of `val` when the associated long-form option is present at the command
     * line.
     * if `flag` is NULL, then `val` is returned by `getopt_long` (see below)
     * when the associated long-form option is found amongst the command-line
     * arguments.
     */
    struct option longopts[] = {
        { "help", no_argument, &help_flag, 1 },
        { "file", optional_argument, NULL, 'f' },
        { "msg", required_argument, NULL, 'm' },
        { 0 }
    };

    /* infinite loop, to be broken when we are done parsing options */
    while (1) {
        /* getopt_long supports GNU-style full-word "long" options in addition
         * to the single-character "short" options which are supported by
         * getopt.
         * the third argument is a collection of supported short-form options.
         * these do not necessarily have to correlate to the long-form options.
         * one colon after an option indicates that it has an argument, two
         * indicates that the argument is optional. order is unimportant.
         */
        opt = getopt_long (argc, argv, "hf::m:", longopts, 0);

        if (opt == -1) {
            /* a return value of -1 indicates that there are no more options */
            break;
        }

        switch (opt) {
        case 'h':
            /* the help_flag and value are specified in the longopts table,
             * which means that when the --help option is specified (in its long
             * form), the help_flag variable will be automatically set.
             * however, the parser for short-form options does not support the
             * automatic setting of flags, so we still need this code to set the
             * help_flag manually when the -h option is specified.
             */
            help_flag = 1;
            break;
        case 'f':
            /* optarg is a global variable in getopt.h. it contains the argument
             * for this option. it is null if there was no argument.
             */
            printf ("outarg: '%s'\n", optarg);
            strncpy (filename, optarg ? optarg : "out.txt", sizeof (filename));
            /* strncpy does not fully guarantee null-termination */
            filename[sizeof (filename) - 1] = '\0';
            break;
        case 'm':
            /* since the argument for this option is required, getopt guarantees
             * that aptarg is non-null.
             */
            strncpy (message, optarg, sizeof (message));
            message[sizeof (message) - 1] = '\0';
```

```
            break;
        case '?':
            /* a return value of '?' indicates that an option was malformed.
             * this could mean that an unrecognized option was given, or that an
             * option which requires an argument did not include an argument.
             */
            usage (stderr, argv[0]);
            return 1;
        default:
            break;
        }
    }

    if (help_flag) {
        usage (stdout, argv[0]);
        return 0;
    }

    if (filename[0]) {
        fp = fopen (filename, "w");
    } else {
        fp = stdout;
    }

    if (!fp) {
        fprintf(stderr, "Failed to open file.\n");
        return 1;
    }

    fprintf (fp, "%s\n", message);

    fclose (fp);
    return 0;
}
```

It can be compiled with `gcc`:

```
gcc example.c -o example
```

It supports three command-line options (`--help`, `--file`, and `--msg`). All have a "short form" as well (`-h`, `-f`, and `-m`). The "file" and "msg" options both accept arguments. If you specify the "msg" option, its argument is required.

Arguments for options are formatted as:

- `--option=value` (for long-form options)
- `-ovalue` or `-o"value"` (for short-form options)