Chapter 7
Gengetopt

7.1 ▪ Command-line options

Most programs presented in this book are expected to be invoked with command-line options. For instance, the program *image-analysis* of Chapter 16 is invoked as

```
./image-analysis  rel_err  infile  outfile
```

The program reads the image file `infile`, performs a wavelet analysis on the image, truncates as much of the insignificant image details as possible while keeping the relative truncation error under the specified `rel_err`, and then writes out the now reduced image into the file `outfile`.

Our *image-analysis* receives and parses the command-line options according to the scheme described in Section 4.7. Parsing three or so command-like options is straightforward, so that's what we do for most of the programs in this book. But when a program calls for much greater number of command-line options, writing code to analyze the command-line becomes a tedious and somewhat uninviting task. In such cases it's preferable to delegate that job to a generic command-line parser utility such as the open source Unix utility, GNU *gengetopt*. We make use of that utility in Chapter 29 where we write programs that apply the neural network methodology to solve various boundary value problems of partial differential equations. Those programs accept quite a few command-line options, such as

```
./demo-square -n 8 -t 15 -T 6 -s 3 -M "outfile.m" -v
```

The purpose of this chapter is to explain how to apply *gengetopt* to manage such command-line options with ease.

7.2 ▪ An overview of *gengetopt*

To apply *gengetopt*, we write a configuration file, let's call it *config.ggo*,[15] that contains a description, in a special syntax (not in C) which we are going to describe, of the intended meanings of a program's command-line options. Then we feed the configuration file to *gengetopt*, as in:

```
gengetopt ... < config.ggo
```

[15] The name of the configuration file can be anything, although it's common to give it a *.ggo* extension.

Listing 7.1: The `--help` option (or `-h` for short) reveals a brief description of the available options.

```
1   $ ./gengetopt-demo --help
2
3   Usage:
4       ./gengetopt-demo   options...
5
6   Options:
7
8     -h, --help              Print help and exit
9     -V, --version           Print version and exit
10    -r, --rows=int          number of rows in matrix
11    -c, --cols=int          number of columns in matrix  (default=`7')
12    -e, --rel-error=float   relative error  (default=`0.12')
13    -o, --outfile=filename  file name for the program's output (no default)
14    -v, --verbose           produce verbose output  (default=off)
```

where the ... are zero or more options (more on that later) that affect *gengetopt*'s behavior.

If all goes well, *gengetopt* writes two C files, *cmdline.[ch]*, which you compile and link with the rest of your program. Courtesy of those files, you now have access to the command-line options from within your program. We will see the details of how that works in the following sections.

Gengetopt is a relatively large program and in this chapter we are not going to cover every aspect of its functionality. Rather, we focus on those aspects of *gengetopt* that are immediately applicable to the kinds of programs that we write in this book. If you need functionality that goes beyond what is found here, you should consult *gengetopt*'s full documentation at

```
https://www.gnu.org/software/gengetopt/gengetopt.html
```

In the rest of this chapter we will work with a short program, *gengetopt-demo.c*, that illustrates *gengetopt*'s functionality. The program does nothing particular—it just reads its mostly meaningless command-line options and prints them out. That's all. Here's how we execute the demo program *gengetopt-demo*:

```
$ ./gengetopt-demo --rows=12 --cols=10 --rel-error=0.05 --outfile="/tmp/zz.m"
    You said: rows=12, cols=10, rel_err=0.05, outfille=/tmp/zz.m
```

The first line is what you type, and the second line is what the program prints.

If we forget what command-line options are available, or what they mean, we may request help by invoking the program with the `--help` option. Then the program prints the list of available options along with a brief description of each. See Listing 7.1 for what that looks like. The `--rows=int` on line 10 in that listing tells us that the `--rows` command-line option takes an `int` (that is, an integer) argument. Similarly, the `-rel-error=float` and `--outfile=filename` command-line options call for a floating point number and a file name (that is, a string) as arguments, respectively.

The single-letter options in the leftmost column in the listing above provide short alternative to their corresponding long versions. They make it possible to enter the previous longish command in a shorter but equivalent form, as:

```
$ ./gengetopt-demo -r 12 -c 10 -e 0.05 -o "/tmp/zz.m"
    You said: rows=12, cols=10, rel_err=0.05, outfille=/tmp/zz.m
```

Command line options that are preceded by two dashes, such as `--rows`, are called *long options*. Command line options that are preceded by a single dash, such as `-r`, are called

short options. Note that we put an equal sign between a long option and its argument, while the argument of a short option is given without an equal sign. In fact, the space between a short option and its argument is optional. Thus, `-r 12` is equivalent to `-r12`.

On a related issue, a long option may be shortened provided that it remains uniquely identifiable. For instance, `--rel-error=0.05` may be abbreviated to `--re=0.05`.

Let us play around a bit more with our program:

```
$ ./gengetopt-demo
./gengetopt-demo: '--rows' ('-r') option required
```

Okay, then let's supply the `--rows` option:

```
$ ./gengetopt-demo -r 4
You said: rows=4, cols=7, rel_err=0.12
```

Where did the `cols=7` and `rel_err=0.12` values come from? They are defaults defined in *config.ggo*. The default values are used when you don't specify `cols` or `rel_err` on the command-line. There is no default value associated with `rows`, that's why the program refuses to run without it. We may override the default values, as in:

```
$ ./gengetopt-demo -r 4 -c 100 --rel=0.005
You said: rows=4, cols=100, rel_err=0.005
```

(Observe that we may mix long and short options in a single command.) As another experiment, let's supply the `outfile` option:

```
$ ./gengetopt-demo -r 4 --outfile="junk"
You said: rows=4, cols=7, rel_err=0.12, outfille=junk
```

Comparing this result with the previous experiments, we see that the `outfile` option has no default value and that's why it has not shown up until now. In a sense, the `outfile` option is like the `rows` option in that neither has a default value. As the same time, the two options are quite different in that omitting the `outfile` option is permitted, while omitting the `rows` option is not. As you may expect, and as we shall see in Section 7.3, these differences stem from the way those options are coded in the *config.ggo* file.

Let us do a final experiment:

```
$ ./gengetopt-demo -r 10 -c 20 -o "junk" -v
--- This is a demo of the gengetopt utility
--- Invoke the program as "./gengetopt-demo --help" for a help message.
    rows=10
    cols=20
    rel_err=0.12
    outfille=junk
--- That's all!
```

The longish output is due to the *flag* `-v` (which may be given as the long option `--verbose`). Unlike the other command-line options that we have seen so far, this option takes no arguments; the program changes its behavior merely as a result of its presence or absence. As we will see in Section 7.5, our *gengetopt-demo.c* is set up so that it outputs more information—is more verbose—when that flag is present.

7.3 ▪ The configuration file

The experiments in the previous section must have given you a general idea of how the the command-line options affect the behavior of the program *gengetopt-demo*. That behavior is dictated by the contents of the configuration file *config.ggo* which is shown in its entirety in Listing 7.2. Let us go through it line by line.

Listing 7.2: The configuration file *config.ggo*.

```
1  package "Gengetopt Demo"
2  version "1.0"
3  usage "\n\t./gengetopt-demo   options..."
4
5  description "Options:"
6
7  option "rows" r "number of rows in matrix"
8          int typestr="int" required
9
10 option "cols" c "number of columns in matrix"
11         int typestr="int" optional default="7"
12
13 option "rel-error" e "relative error"
14         double typestr="float" optional default="0.12"
15
16 option "outfile" o "file name for the program's output (no default)"
17         string typestr="filename" optional
18
19 option "verbose" v "produce verbose output"
20     flag off
```

Lines 1 and 2: We have no use for `package` and `version`, but their presence is required by *gengetopt*, so there they are.

Lines 3 and 5: These are responsible for what we see on lines 3 through 6 in Listing 7.1.

Lines 7–8: Here we introduce the long option `rows` and the associated short option `r`. Line 7 provides a brief description of the purpose of those options. That description shows up on line 10 in Listing 7.1.

The `int` on line Line 8 indicates that the option `row` takes an integer for argument. If you omit that argument, or if supply a non-integer, the program will refuse to run.

The `typestr="int"` has only a cosmetic effect; it manifests itself as `--row=int` on on line 10 in Listing 7.1. Without the `typestr="int"`, that line would have come out as `--row=INT` which is perhaps a bit gaudy. You may specify anything else that you prefer over `typestr="int"`. Try, for instance, `typestr="#"`.

Finally, the `required` on line 8 indicates that specifying the `rows` option on the command-line is mandatory. In the previous section we saw that in the absence of that option the program refuses to run.

Lines 10–11: Here we introduce the long option `cols` and the associated short option `c`. This block is quite similar to that of the preceding `rows` option, so here we address the differences only.

The `optional default="7"` on line 11 indicates that providing a `cols` option on the command-line is optional. If that option is omitted, `cols` takes on the default value of 7.

Lines 13–14: This is very much like the previous blocks, The only new feature is that the argument is of the type **`double`** rather than **`int`**.

Lines 16–17: Here we introduce the long option `outfile` and the associated short option `o`. The `string` on line 17 indicates that the argument of that option will be

Listing 7.3: A *Makefile* for *gengetopt-demo*.

```
1  OFILES = cmdline.o gengetopt-demo.o
2  CFLAGS = -Wall -pedantic -std=c99 -O2
3  TARGET = gengetopt-demo
4
5  $(TARGET): $(OFILES)
6      $(CC) $(OFILES) -o $@
7
8  gengetopt-demo.o: gengetopt-demo.c cmdline.h
9
10 cmdline.c cmdline.h: config.ggo
11     gengetopt < config.ggo
12
13 clean:
14     /bin/rm -f $(OFILES) $(TARGET) cmdline.[ch]
```

interpreted as as C string, or to be precise, as a pointer to a C string. The `optional` indicates that providing this command-line option is not mandatory, but in its absence, the value of `outfile` will be `NULL` since no default value is specified.

The `typestr="filename"` has only a cosmetic effect; it manifests itself as the `--outfile=filename` on line 13 of Listing 7.1. Without the `typestr="filename"`, that line would have been printed as `--outfile=STRING` which is somewhat less expressive.

19–20: The command-line option `--verbose` (or its short version, `-v`) is a *flag*. The value of this option is captured as 0 or 1 within the program *gengetopt-demo.c*. The `off` on line 20 indicates that the flag will be off (will have the value 0) by default. Giving the `--verbose` option on the command-line turns the flag on.

Aside: If we change line 20 to `flag on`, then the flag will be on (will have the value 1) by default. Giving the `--verbose` option on the command-line will turn the flag off.

7.4 ▪ From *config.ggo* to *cmdline.[ch]*

The configuration file *config.ggo* which we examined in the previous section is certainly *not* a C program. The *gengetopt* utility reads that configuration file and writes the C files *cmdline.[ch]*. In a Unix system you effect that by entering

```
$ gengetopt < config.ggo
```

on the command line. That is quite workable in the case of a brief demo program such as the one under consideration. Realistic programs, however, tend to be significantly more complex and are hardly ever compiled without a corresponding *Makefile*. Let us, therefore, see how a Makefile can be constructed in our case. You will need to consult Chapter 6 if you need help with `make`.

Listing 7.3 shows a suggested *Makefile*. Let us examine its details.

[

Lines 1–3: Here we declare the program's object files, *cmdline.o* and *gengetopt-demo.o*, the compiler options, `CFLAGS`, and the name of the desired executable, *gengetopt-demo*.

Lines 5–6: The executable is produced through linking the object files.

Listing 7.4: An outline of the file *gengetopt-demo.c*, showing its basic structure.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "cmdline.h"
4
5  int main(int argc, char **argv)
6  {
7      struct gengetopt_args_info args_info;
8
9      if (cmdline_parser(argc, argv, &args_info) ≠ 0) {
10         fprintf(stderr, "Command line parser failed (why)?\n");
11         return EXIT_FAILURE;
12     }
13
14     int rows = args_info.rows_arg;
15     int cols = args_info.cols_arg;
16     double rel_err = args_info.rel_error_arg;
17     char *outfile = args_info.outfile_arg;
18     int verbose = args_info.verbose_given;
19
20     ... make use of the values of rows, cols, etc. ---
21
22     cmdline_parser_free(&args_info);
23
24     return EXIT_SUCCESS;
25 }
```

Line 8: The object file *gengetopt-demo.o* depends directly on *gengetopt-demo.c*, and indirectly on *cmdline.h* since, as we will see in the next section, *gengetopt-demo.c* `#include`s the file *cmdline.h*.

Lines 10–11: Here we are telling `make` that the files *cmdline.[ch]* are produced by feeding *config.ggo* to *gengetopt*.

Lines 13–14: The command `make clean` executes the Unix command on line 14 to remove the object files, the target file, and the files *cmdline.[ch]*, returning the current directory/folder to its pristine state.

7.5 ▪ The file *gengetopt-demo.c*

We have seen how *gengetopt-demo* behaves, and we have seen how that behavior is encoded in *config.ggo*. Now we examine the contents of *gengetopt-demo.c* to see how it accesses its command-line arguments.

7.5.1 ▪ An outline of *gengetopt-demo.c*

Before looking at the actual *gengetopt-demo.c*, let's look at the outline shown in Listing 7.4 to get a sense of the program's structure. This outline is common to all applications of *gengetopt*.

Line 7 in that listing declares a `args_info` structure which is to hold the data extracted from the command line.

Lines 9–12 initialize the *gengetopt* parser. In case of trouble (computer out of memory?) we print a message and exit.

Lines 14–18 assign names meaningful in the current context to the generic names provided by *gengetopt*. For instance, what *gengetopt* calls `args_info.rel_error_arg`, we call `rel_err`. Let us note that

1. The name `rel_error_arg` is constructed by *gengetopt* through modifying what we called `rel-error` in *config.ggo* (see line 13 of Listing 7.2 on page 46) by changing the hyphen to an underscore in order to make it into a legal identifier in C.

2. *Gengetopt* constructs the names for the arguments of the command-line options by adding the `_arg` suffix, as in `rel_error_arg`. But our `verbose` command-line option is a *flags*, that is, it takes no arguments, and therefore `verbose_arg` would be meaningless. In line 18 we see that *gengetopt* signals the presence of the `verbose` flag through the name `args_info.verbose_given`.

Line 20 is where our program does its work. We will see an instance of what goes there in the full listing of *gengetopt-demo.c*. Once that work is complete, we free the memory allocated by *gengetopt* (line 22) and exit.

7.5.2 ▪ The complete *gengetopt-demo.c*

The complete contents of *gengetopt-demo.c* are shown in Listing 7.5 This a strict superset of the outline in Listing 7.4. The lines 20 through 37 are new and their meanings should be self-evident. The code distinguishes between the cases where the `verbose` flag is given versus when it is not. In the former case the program prints detailed information about what it has read from its command-line. In the latter case only a brief summary is printed.

Equipped with *config.ggo* from Listing 7.2, *Makefile* from Listing 7.3, and *gengetopt-demo.c* from Listing 7.5, you should be able to compile the program and run the tests that were demonstrated in Section 7.2.

Listing 7.5: The complete *gengetopt-demo.c*.

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

int main(int argc, char **argv)
{
    struct gengetopt_args_info args_info;

    if (cmdline_parser(argc, argv, &args_info) ≠ 0) {
        fprintf(stderr, "Command line parser failed (why)?\n");
        return EXIT_FAILURE;
    }

    int rows = args_info.rows_arg;
    int cols = args_info.cols_arg;
    double rel_err = args_info.rel_error_arg;
    char *outfile = args_info.outfile_arg;
    int verbose = args_info.verbose_given;

    if (verbose) {
        printf("--- This is a demo of the gengetopt utility\n");
        printf("--- Invoke the program as \"%s --help\""
                " for a help message.\n", argv[0]);
        printf("\trows=%d\n", rows);
        printf("\tcols=%d\n", cols);
        printf("\trel_err=%g\n", rel_err);
        if (outfile ≠ NULL)
            printf("\toutfille=%s\n", outfile);
        printf("--- That's all!\n");
    } else {
        printf("\tYou said: rows=%d, cols=%d, rel_err=%g",
                rows, cols, rel_err);
        if (outfile ≠ NULL)
            printf(", outfille=%s\n", outfile);
        else
            putchar('\n');
    }

    cmdline_parser_free(&args_info);

    return EXIT_SUCCESS;
}
```