Program 2: getopt_long()

CSCI 4547 / 6647 Systems Programming
Fall 2020

1 Goals

1. To develop a command language for a utility that combines features from find and grep.
2. To use `getopt_long()` to process options.
3. To learn or review some parts of C++ that will be needed in this course.

2 The Project

The command that you develop will be named `sniff`. It will search your disk for files that contain one or more of a set of words that will be given on the command line. Design a command-line language to meet these specifications:

- There must be short switches, long switches, and a switch with an argument.
- Some but not all of the short switches must have synonymous long switches, and vice versa.
- Allow both case-insensitive searches and case-sensitive searches (the default).
- Allow a search to start in the current working directory (the default) or in a subdirectory , for which a pathname is supplied.
- The default is screen output, but provide a switch with a filename argument to use for output.
- Provide a switch to allow verbose output, that is, print the name of every file and directory that is opened. This information can be crucial during development and debugging.
- Provide a help switch that will give info about how to use sniff.
- The command line should provide a single quoted string of search words, separated by spaces.

Write a document that describes your command language and explains how to use it.

3 Instructions

Write a program consisting of a main function and the class Params (described below). In your program, use `getopt_long()` to parse a command line for sniff. Strive for concise code.

In your main function, Main should accept argc and argv, which is an array of cstrings. You need to convert some of them to C++ strings. The easiest way is to just call the C++ string constructor with the argv input; the prototype is `string (const char* s);` (Refer to the stringstr.cpp demo.)

In main(), declare an instance of Params and pass argc and argv to its constructor. When construction is finished, call Params::print() to display the params.

The Params Class. Data members of the class should include:

- The pathname of the starting directory, a C-style string. You will `chdir` to this directory.
- The name of an output file, a C-style string.
- An ofstream for your output file. Open this file when the name is captured.

- Boolean variables for all the switches. (The initial values should all be false.)
- For the search words, argv gives you a c-string containing the words. You need to convert that to a C++ string and store it. (This will be unpacked into a vector of words in Program3.)

Function members of the class should include:

- The Params constructor, with two parameters: argc and argv. Process the command-line arguments using `getopt_long()` and initialize the data members to the settings that you find on the command line. Open any files.
- print( ): Print all data members (except the ofstream itself) to the open output stream. Format the display neatly, like the one shown below. Here is some sample output:

```
Command: sniff --verbose -o found.txt  --dir ~/A_UNH/Teaching  "CSCI"
    Start at: ~/A_UNH/cs6647
    Output file name:  test.txt
    Verbose? Yes
     Ignore case?  No
```

- void usage(): print a usage comment that explains the format of the `sniff` command and lists the legal switches.

Testing. Make a special test directory for this project on your disk and use it consistently.

- Each week, as you develop another part of this application, you will add some files or directories to it.
- For now, select a theme and two to five search words related to that theme. For example, theme: election. search words: Trump, Biden, president, candidate, election.
- In your test directory, create several short text files that contain different subsets of your search words, including files with zero, one, and more of these words. Add a soft link, a hard link, a subdirectory, and a few files that are not text files.
- Test all of the command-line options and capture the results from all tests in one output file, using append mode.

Submission. Submit a set of related things, all zipped together.

1. Your main program and Params class (three files, one.hpp for Params and two .cpp files for Params and main).
2. A screenshot that shows the files in the test directory you created followed by your call on em sniff.
3. A file containing the output from all your test runs.