Chapter 3 Practice Exercises

Disclaimer: These practice questions are not official and may not have been vetted for course material. You may use these for practice, but you should prioritize official resources from current staff for a more accurate depiction of what you need to know for assignments and exams.

1. Consider the following statement on the command line. What is the value of `argc`?

```
./letter -q -c -l -p -o M -b apple -e orange < dictionary.txt
```

A) 7
B) 10
C) 11
D) 12
E) 13

2. You are writing a command line program that can be used to fetch historical weather conditions for locations in the United States. When the user runs the executable for this program (`./weather`), they are able to pass in a numerical zip code using the `--zip` flag (short form `-z`), a numerical date in YYYYMMDD format using the `--date` flag (short form `-d`), and an optional `--verbose` flag that can be used to display the results in verbose mode (short form `-v`). The verbose flag does not take in any subsequent arguments. An example is shown below (this displays the weather details for zip code 48109 on May 9, 2023 in verbose mode):

```
./weather --zip 48109 --date 20230509 --verbose
```

Which of the following `option` objects for `getopt_long()` best depicts each of these command line options?

A)
```
{"zip", no_argument, nullptr, 'z'},
{"date", no_argument, nullptr, 'd'},
{"verbose", optional_argument, nullptr, 'v'}
```

B)
```
{"zip", required_argument, nullptr, 'z'},
{"date", required_argument, nullptr, 'd'},
{"verbose", no_argument, nullptr, 'v'}
```

C)
```
{"zip", required_argument, nullptr, 'z'},
{"date", required_argument, nullptr, 'd'},
{"verbose", optional_argument, nullptr, 'v'}
```

D)
```
{"zip", optional_argument, nullptr, 'z'},
{"date", optional_argument, nullptr, 'd'},
{"verbose", required_argument, nullptr, 'v'}
```

E) None of the above

3. Consider the same program as described in question 2. Which of the following strings is the correct short options string for use with `getopt_long()`?
A) `zdv`
B) `z:d:v:`
C) `zd:v`
D) `zdv:`
E) `z:d:v`

4. Your friend is implementing a command line program that can be used to rank users on a class forum by their number of posts. This program accepts two options on the command line: `--rank` (short form `-r`) followed by a required integer argument, and `--help` (short form `-h`) followed by no argument. This is the `long_opts[]` array that they came up with:

```
1  static struct option long_opts[] = {
2    { "rank", required_argument, nullptr, 'r' },
3    { "help", no_argument, nullptr, 'h' }
4  };
```

Does this work as intended? If not, what is the issue?
A) No, the `long_opts` array cannot be declared as `static`
B) No, the fourth member of the `rank` option on line 2 should be `"r:"` instead of `'r'`
C) No, the last item of the `long_opts` must be an option with all its members set to zero
D) No, for more than one of the reasons above
E) Yes, this `long_opts` array works as intended

5. What is the purpose of the `optarg` variable in the `getopt_long()` function?
   A) `optarg` stores the value of the short options string
   B) `optarg` keeps track of how many arguments there are to process
   C) `optarg` stores the value of the supplied argument for options that accept arguments
   D) `optarg` holds the integer values corresponding to the short options present in the command line
   E) None of the above

6. Consider the following `long_opts` array:

```
static struct option long_opts[] = {
  { "apple", no_argument, nullptr, 'a' },
  { "banana", required_argument, nullptr, 'b' },
  { "orange", no_argument, nullptr, 'o' },
  { "grape", required_argument, nullptr, 'g' },
  { "pear", no_argument, nullptr, 'p' },
  { nullptr, 0, nullptr, '\0' }
};
```

   Which of the following commands would **NOT** be valid? Assume the exectuable is named `exec`.
   A) `./exec -abg 12 --orange < fruits.txt`
   B) `./exec --apple -pb fruits.txt < fruits.txt`
   C) `./exec -o -g 13 -aob pineapple > fruits.txt`
   D) `./exec -pao --grape 281 --banana grape`
   E) More than one of the above

7. Consider the following `switch` statement:

```
int32_t num;
std::cin >> num;
switch (num) {
  case 1:
    std::string str = "yes";
    std::cout << str << '\n';
    break;
  case 0:
    std::string str = "no";
    std::cout << str << '\n';
    break;
  default:
    break;
}
```

   Which of the following statements is **TRUE** regarding this code?
   A) The code compiles, and prints "yes" if 1 is passed into standard input, and "no" if 0 is passed into standard input
   B) The code compiles, but nothing gets printed if 1 or 0 is passed into standard input
   C) The code does not compile, since it is impossible to declare variables within a switch statement
   D) The code does not compile, since variables declared in one case are still visible in another case since they are part of the same scope
   E) The code does not compile, since cases must be defined in ascending order

8. Consider the following code, which is partially completed. There are two valid command line options. The `'l'` or `"level"` option has a required argument in the form of an integer, and the value of this argument is assigned to the variable `initial_level`. The `'h'` or `"hard"` option turns `hard_mode` on, setting the bool to `true`. The `'h'` option has no argument. Assuming all commands are valid, complete the code below (you may use the `atoi()` method to convert a valid C-string into an integral number):

```
class Game {
  int initial_level = 0;
  bool hard_mode = false;
public:
  void get_options(int argc, char** argv);
  /* ... other members ... */
};

void Game::get_options(int argc, char** argv) {
  int option = 0;
  int index = 0;
  // TODO: Fill out the option struct
  static struct option long_opts[] = {










  };

  // TODO: Complete the while loop
  while ((option = getopt_long(           ,            ,            , long_opts, &index)) != -1) {
    switch (option) {
      case 'l':









      case 'h':









    } // switch
  } // while
} // get_options()
```

Chapter 3 Exercise Solutions

1. **The correct answer is (C).** The input redirection components of `"<"` and `"dictionary.txt"` do not contribute to `argc` and `argv`. Thus, `argc` is the number of other items in the command line, which comes out to 11.
2. **The correct answer is (B).** Both `zip` and `date` require additional arguments to be provided if they are specified, so these two command line options should be defined with a `required_argument`. On the other hand, `verbose` does not take in any subsequent arguments if it is specified, so it should be defined with `no_argument`. Notice that `optional_argument` indicates that an argument is optional *if the option is specified*, which does not apply here.
3. **The correct answer is (E).** Options that require arguments (in this case, `z` and `d`) should be followed by a colon.
4. **The correct answer is (C).** The last item of the `long_opts` array should be an option with all zeros, so that `getopt_long()` knows when there are no more option choices remaining. The options array can be defined as `static`, and the additional colon is needed for the short options string within the `getopt_long()` function call, but not in the option declaration itself.
5. **The correct answer is (C).** For command line options that accept arguments, `optarg` is a C-string that stores the values of these arguments.
6. **The correct answer is (A).** If `banana/b` or `grape/g` is specified on the command line, they must be followed with an additional argument. If `apple/a`, `orange/o`, or `pear/p` is specified, they must be followed with no additional argument. The only command for which this is not true is (A), since `b` is specified without an argument after it (the `12` is the argument for `g`). Note that there is no restriction on including an option more than once for `getopt_long()`.
7. **The correct answer is (D).** The `switch` statement here does not compile because the newly created variables are not scoped properly — the lifetime of the string instantiated on line 5 extends past its specific case, which is not allowed. To fix this, curly braces can be added to each case, as shown:

```cpp
int32_t num;
std::cin >> num;
switch (num) {
  case 1:
  {
    std::string str = "yes";
    std::cout << str << '\n';
    break;
  }
  case 0:
  {
    std::string str = "no";
    std::cout << str << '\n';
    break;
  }
  default:
  {
    break;
  }
}
```

8. The completed code is shown below. The `'l'` case sets `initial_level` to the value of `optarg` as an integer, and the `'h'` case sets `hard_mode` to true.

```cpp
class Game {
  int initial_level = 0;
  bool hard_mode = false;
public:
  void get_options(int argc, char** argv);
  /* ... other members ... */
};

void Game::get_options(int argc, char** argv) {
  int option = 0;
  int index = 0;
  static struct option long_opts[] = {
    { "level", required_argument, nullptr, 'l' },
    { "hard", no_argument, nullptr, 'h' },
    { nullptr, 0, nullptr, '\0' }
  };

  while ((option = getopt_long(argc, argv, "l:h", long_opts, &index)) != -1) {
    switch (option) {
      case 'l':
        initial_level = atoi(optarg);
        break;
      case 'h':
        hard_mode = true;
        break;
    } // switch
  } // while
} // get_options()
```