CMPT 214: Programming Principles and Practice
Term 1 2016-17
Lab 6 - data types, software development practices, and more UNIX

At the beginning of your lab period, the lab instructor will give an example of calculating function, branch, and condition coverage for a program. Recall that branch coverage is also called path coverage.

Answer each of the questions below. For questions 3 and 4, you will need to download the auxiliary file `L6Q4.txt`; look for the link on the moodle pages for the course. For all questions involving the use of LINUX/UNIX commands, place the command or pipeline you used along with the resulting output (i.e. copy-and-paste from your terminal window) in a file called `lab6.txt`. However, do not include extraneous or superfluous commands or output; only include content relevant and essential to the specified task. Then, with a text editor, add to `lab6.txt` your solutions to question 1(c) and all the parts of questions 2 and 4. Also with the text editor add text and identifying information to clearly distinguish which commands/output/code/description correspond to each task/question. This lab is out of a total of 14 marks, with each question or question major part (1a, 1b, 3, etc.) being worth one mark except for 4(b) and 4(c), which are each worth 2 marks. Marks may be docked for extraneous, irrelevant, or superfluous content or for not following directions. Your submission is due at 11:55 p.m. on Thursday, October 20. Note that this lab specification is 3 pages in length.

This lab can be completed on either tuxworld or one of the Macintosh computers in S311 and S315. However, clearly identify in your lab submission which (type of) machine you used. The answers to some of the lab questions vary slightly depending on the (type of) machine used, and therefore the marker needs to know which it was.

1. (a) Create a procedural C++ program called `datatype_sizes.cc` that will output the sizes in bytes of the datatypes `long int`, `float`, `double` and `float *`. Make your program simple and and its logic straightforward. For example, the program logic can consist solely of four `printf()` statements. Upload your program to moodle as part of your Lab 06 submission when complete. With the exception of warnings about parameters `argc` and `argv` of function `main()` not being used, your program must compile without errors or warnings when the `-Wall` and `-Wextra` options are given to the C++ compiler.
   Hint: use the C/C++ function `sizeof()`. The return type of `sizeof()` is `size_t`. To have `printf(3)` output a value of type `size_t`, use the "`%zu`" or "`%zd`" format specification. The program example from class on October 13 would be a very good starting point for your program. Note: overly and unnecessarily complex programs may be docked marks.

   (b) Compile your C++ program two ways: first with the option "`-m32`" and then with the option "`-m64`". These options specify 32- and 64-bit architecture, respectively. Name the resultant executables `32bit_sizes` and `64bit_sizes`, respectively. Show a log of your compilations in `lab6.txt`. Have the C++ compiler perform extra checks for potentially problematic situations by using the `-Wall` and `-Wextra` options. You can also use `-Wpedantic` if you wish.

(c) Run each of your executables in part (b) (include a log of this in `lab6.txt`), and then write a sentence or two comparing their outputs; that is, comparing the size of datatypes used by the C/C++ compiler when compiling for the two different architectures (32-bit or 64-bit).

(d) Use a UNIX command to determine the file types of `32bit_sizes` and `64bit_sizes`; i.e. use a UNIX command to determine the kind of information within the two files.

(e) Use `uname(1)` with an appropriate option to produce, on the standard output, the "machine hardware name" of the machine on which you completed the previous parts to this question. Include in your lab submission a log of the command you used, and its output.

2. (a) For the following pair of data definitions, state which one conforms to the programming guidelines given in lecture. Place your answer (either "i" or "ii") in `lab6.txt`.

i.
```
// A food is one of the four food groups.
typedef enum { MEAT, DAIRY, VEGETABLE, GRAIN } food_group;
// ... ...(food_group food, ...) {
// ...
// switch (food) {
//     case MEAT: ...; break;
//     case DAIRY: ...; break;
//     ...
//     default: assert(false); break;
// }
```

ii.
```
// A food is one of the four food groups.
typedef enum { MEAT, DAIRY, VEGETABLE, GRAIN } food_group;
```

(b) For the following pair of module templates, state which one conforms to the programming guidelines given in lecture. Place your answer (either "i" or "ii") in `lab6.txt`.

i.
```
// For a given person, determine which food group they eat the least.
food_group least_eaten(person *p) {
     return MEAT;
}
```

ii.
```
// least_eaten :  For a given person pointed to by p,
// determine which food group they eat the least.
// In:
//   p != NULL
// Out:
//   a value from enumerated type food_group
food_group least_eaten(person *p) {
     return MEAT;
}
```

3. Use the `pr` command to produce, on the standard output, a listing of `L6Q4.txt` that has the lines numbered consecutively.
Hint: you will need two options to `pr` to get the proper output, one of which is `-t`. You will need to determine the other option.

4. Consider the fragment of code found in `L6Q4.txt`. Suppose that this code was tested using the test cases (i.e. values of `myint`) 0, 5, -8, and -100. Answer the following questions in `lab6.txt`.

(a) Evaluate the function coverage of these test cases. In other words, what proportion of the functions are actually called in at least one of the test cases? Only include in your analysis `add1()`, `substract1()`, `double_it()`, and `halve_it()`—do not consider `scanf()` and `main()`.

(b) Evaluate the path coverage of these test cases. In other words, to what extent has every possible path (of execution) through the program been executed? Expression your answer as a proportion.

(c) Evaluate the condition coverage of these test cases. In other words, what proportion of the conditions in `if` statements evaluate to `true` in at least one of the test cases and to `false` in at least one of the test cases? That is, to be counted as being "covered", a condition must evaluate to both `true` and to `false` as part of the testing.

(d) Given the results of parts (a), (b), and (c), comment on the adequacy of the four test cases (0, 5, -8, and -100) in terms of providing a thorough test of `L6Q4.txt`.