Programming Project Submission Guidelines

CS 273 R. Brown March 11, 2018

Programming project submissions consist of the following parts, all submitted via `stogit`:

(1) a cover page `COVER.md` as described below; for multi-stage projects, each stage submission should update the cover page;
(2) your source code, including `Makefile` if available;
(3) a file `README.md` containing any external documentation, including information on any extra features.

(1) Cover Page

Include a cover page file `COVER.md` at the top level of your project directory (e.g., `~/OS/pp-shell/COVER.md`).

- Your cover page should include the following form of the pledge.

> I pledge my honor that I have neither given nor received assistance on this programming project, except as explicitly stated on this page, and that I have seen no dishonest work.
>
> Signed _____
>
> ☐ I have intentionally not signed the pledge (*check only if appropriate*)

  - *To indicate signing the pledge*, type your name and enter either `[ ]` or a box near the last line of the pledge statement. (`stogit` authentication confirms your signature.)
  - *To indicate* **not** *signing the pledge*, do **not** type your name, and enter either X or a checkmark near the last line of the pledge statement.
- After the pledge enter a statement of help received (on that stage, for a multi-stage project), for example,

```
== Help received ==
I talked with Jane Doe about how to use execve()
```

or

```
== Help received ==
I received no help on this phase of this assignment
```

You are not expected to do it all alone, but you are expected to cite your sources honestly. Joint projects are not permitted without special approval in advance.

(3) README file

Submit a file `README.md` in the top level of your project directory.

- Include any external documentation for your program, including instructions for building the code and running the program. Use the following line as a header for your external documentation:

```
== External Documentation ==
```

- List any extra features beyond a basic assignment in order to demonstrate your insight beyond the basic material and improve your grade. Some significant extra features are suggested in each programming-project assignment sheet; you may also think of some ideas of your own. Use the following line as a header for your extra feature list:

```
== Extra Features ==
```

Note: Plan to complete the basic assignment before tackling any extra features. This is a matter of self-protection: it is better to submit a well-done basic assignment on time than a late assignment with extras. Extra features will be judged on the basis of appropriateness, difficulty, creativity and originality, use of the programming language, programming style and documentation. Point out your extra features clearly in README.md, or they will probably go unnoticed.

Late Submissions

Penalties for late projects will take into account any extenuating circumstances and the number of days late. Please let me know as soon as you know you will realize your project will be late.

Extra features on late submissions will ordinarily be disregarded—it is better to hand in the basic assignment on time then to turn in a late assignment with extras.

Documentation Standard

Projects should include documentation as follows.

- Include a comment block at the beginning of each file briefly stating (1) the purpose of the code (2) identifying information such as author and date.

- Include a comment block near the heading of the most important functions that includes a specification. Specifications provide concise and unambiguous descriptions of any arguments, return value and "out" parameters, and describe the logical effects of the function, e.g., via preconditions and postconditions. Use a consistent and easily digested format for specifications.

  Less important functions, such as routine auxiliary functions, may be described in a summary line or two.

- Give concise data invariants for all important global and local variables and data structures. A *data invariant* states the logical properties of a variable or data structure that remain true whenever that variable or structure is used.

- Include invariant assertions and loop invariants (iteration invariants) to outline the logical progression of computations. An *invariant assertion* describes the logical state of the computation at a given point. An *iteration invariant* for a tail-recursive function is an invariant assertion that is true at the beginning and ending of every call of that function. Likewise, a *loop invariant* is an invariant assertion that is true at the beginning and end of *every* iteration of a loop (for imperative languages).