Programming Assignments

1 General Information

1.1 Software Requirements

Detailed specifications are typically provided for the software to be developed for each assignment problem. Such specifications may include, amongst other things, requirements relating to:

- the organization of the source code into files (e.g., file names including case, file contents, etc.);
- data formats for program input and output;
- user-interface behavior; and
- programming interfaces.

It is absolutely critical that all of these specifications be met **exactly** by the submitted code. Failure to meet all specifications exactly will likely prevent the marker from fully testing (i.e., compiling, linking, and running) the submitted code. If the submitted code cannot be fully tested (or cannot be tested at all), marks will be deducted accordingly, and **the penalty may potentially be very significant** (e.g., the penalty could easily result in a failing grade).

The software should not crash under any circumstances (even if the program input is invalid). Do not assume that the program input is valid. Always handle errors gracefully. Each program should terminate immediately upon completing its task. When a program terminates, an integer exit status is returned to the operating system indicating whether the program completed successfully or terminated abnormally (e.g., due to an error). The exit status of a program can be set to the integer value `n` by either terminating the program by calling `std::exit(n)` or returning the value `n` from the function `main`. Unless explicitly indicated otherwise, all programs should follow the convention of setting their exit status to `0` upon success and `1` upon failure.

1.2 Assignment Submission

All programming assignments in the course are to be submitted via GitHub Classroom (`https://classroom.github.com`). For this reason, each student in the class is required to obtain their own personal GitHub account. For information on how to obtain a GitHub account, refer to Section 1.5. GitHub Classroom provides a means for creating private Git repositories for students that can be used for assignment submission. For each assignment, the instructor will send an assignment invitation URL (via email) to all of the students in the class. This URL can then be used by each student to gain access to their own private Git repository for assignment submission. For more information on how to accept an assignment invitation, see Section 1.6.

The contents of a Git repository for an assignment submission are not arbitrary, and must be organized in very specific manner. A detailed description of how the contents of a repository must be organized is provided later in Section 1.3.

For each assignment, each student is responsible for ensuring that their Git repository contains their completed assignment submission prior to the submission deadline. At the time of the submission deadline, the marker (or instructor) will create a snapshot of each of the student assignment repositories either by cloning the repository or tagging the repository with a cryptographically-signed tag. These repository snapshots will then be used for marking. Any attempt by a student to tamper with (e.g., delete) any repository tags added by the marker (or instructor) will be deemed an act of academic fraud (i.e., cheating) and will be handled as such. Assignment resubmissions are not permitted. So, students should be careful to ensure that their repository contains the desired contents before the submission deadline. For the policy on late and incomplete assignments, refer to Section 1.10.3.

Prior to the submission deadline, the student is required to ensure that their repository passes some basic sanity checks performed by the `assignment_precheck` command. The submission of an assignment that does not pass these checks will result in the assignment not being graded and a mark of zero being awarded. For more details about the `assignment_precheck` command, refer to Section 1.4.

1.3 Git Repository Organization

The files in the Git repository for an assignment must be organized in a very specific manner. In what follows, this organization is described in detail.

Unless explicitly indicated otherwise, all of the files contained in the Git repository should be in the top-level directory of the repository (i.e., the repository should not contain any subdirectories). Unless otherwise noted, a single CMakeLists file called `CMakeLists.txt` should be provided for building the programs in the assignment. This CMakeLists file should provide a target for each individual program in the assignment with the same name as the program. If any problems require written responses, the responses should be placed either in a file called `README.txt` in plaintext format or in a file called `README.pdf` in PDF format. Use either the plaintext or PDF format exclusively; do not use both. Questions that require an answer in English (as opposed to source code) must be answered in complete sentences (in coherent English). Since file names are case sensitive on Linux/UNIX systems, it is important that the correct case (i.e., uppercase versus lowercase) be used in file names. Only include the files that are required for the assignment. For example, do not include backup/old versions of files. If a text editor is used that automatically creates backup versions of files, one must be careful not to accidentally submit these backup files.

For identification purposes, the top-level directory of the repository must contain a file named `IDENTIFICATION.txt`. This file must contain the following items in order, each on a separate line:

1. the keyword `name` followed by a space and then the student's full name enclosed in double quotes;
2. the keyword `student_id` followed by a space and then the student's student ID enclosed in double quotes;
3. the keyword `email` followed by a space and then the student's email address enclosed in double quotes;
4. the keyword `section` followed by a space and then the student's tutorial section (e.g., T01 or T02) enclosed in double quotes; and
5. the keyword `assignment` followed by a space and then the assignment ID enclosed in double quotes.

The assignment ID for each assignment is given along with the other information for the assignment. To further illustrate the format of the `IDENTIFICATION.txt` file, we now consider a specific example. For an assignment with the assignment ID `basics` to be submitted by a student in tutorial section T01 who is named Jane Doe with student ID V00123456 and email address `jdoe@uvic.ca`, the `IDENTIFICATION.txt` file should have the following contents:

```
name "Jane Doe"
student_id "V00123456"
email "jdoe@uvic.ca"
section "T01"
assignment "basics"
```

Repository tag and branch names that begin with one or more underscore characters are reserved for use by the course instructor and teaching assistants. It is critically important that students not create (or modify or delete) tags or branches with such names.

It is imperative that the specification for how to organize a repository be followed **exactly**. Failure to do so could result in **a very significant mark penalty or a mark of zero being awarded**.

1.4 The `assignment_precheck` Command

To eliminate the possibility of many basic mistakes in the assignment submission process, a program is provided that performs basic sanity checking on the assignment submission in a Git repository. To perform a basic sanity check on the repository with the URL *url*, use the command:

`assignment_precheck` *url*

This command will check the files associated with the most recent commit on the `master` branch of the repository with URL *url*. If, for some reason, you would like to check the files associated with a different commit (i.e., different from the most recent commit on the `master` branch), a particular branch/commit can be specified with the `-b` option, which takes a branch (or commit) as an argument.

For illustrative purposes, we now consider a few specific examples of using the `assignment_precheck` command. To check the files in the most recent commit of the `master` branch of the repository with URL `https://github.com/uvic-elec486/repo.git`, use the command:

```
assignment_precheck https://github.com/uvic-elec486/repo.git
```

To check the files in the most recent commit of the `experimental` branch of the repository with URL `https://github.com/uvic-elec486/repo.git`, use the command:

```
assignment_precheck -b experimental https://github.com/uvic-elec486/repo.git
```

Two types of checking are performed by the `assignment_precheck` command. First, the command ensures that the files in the repository are organized correctly. This process is referred to as validation. If an assignment submission fails to pass the validation check, the assignment will not be graded and **a mark of zero will be assigned**. Second, the command attempts to configure and build the code exactly as submitted by the student. If the configure or build operation fails for an assignment submission, the submission will still be accepted for marking but the submission will probably be severely penalized (since the code cannot be tested in such circumstances). Consequently, it is advisable to ensure that the configure and build operations pass.

1.5 Obtaining a GitHub Account

To create a GitHub account, complete the account creation form at the following URL:

`https://github.com/join`

During the registration process, you will be asked to provide an email address to be associated with the new GitHub account. It is strongly recommended that you use your UVic email address, as this will allow you to more easily obtain a discounted student account (as described below).

Normally, GitHub users must pay a subscription fee in order to be permitted to create private repositories (i.e., repositories that are not accessible to the public). GitHub, however, offers discounted accounts for students that allow the creation of free private repositories. You should request a student discount by following the instructions at the following URL:

`https://education.github.com/discount_requests/new`

If you associated your UVic email address with your GitHub account when you created it, the approval process for a student discount will likely be easier, as GitHub can more easily confirm that you are affiliated with an educational institution.

1.6 Accepting an Assignment Invitation

Upon receiving an assignment invitation URL, you can gain access to a new repository to be used for the assignment as follows. First, open the invitation URL using your web browser. You will then be prompted for your GitHub credentials in order to login to GitHub. Upon login, you will be asked to accept the new assignment, which you should accept. After accepting the assignment, you should immediately be given access to a new repository (to be used for the assignment). This new repository should appear under the list of repositories for your account on your GitHub home page (i.e., `https://github.com`) when you are logged in. You should also receive an email indicating that you have been subscribed to the new repository. It may take several minutes (or longer) for this notification email to be received.

1.7 Posting of Assignment Solutions

The posting of assignment solutions in any forum to which other students might have access is forbidden (as this would implicitly encourage students to plagiarize the posted solutions). (A public GitHub repository would be an example of such a forum.) **Any student who either posts their assignment solutions or allows their solutions to be posted by another party will be deemed to have committed an act of cheating, and will be dealt with accordingly.** For this reason, students should exercise great care to ensure that their assignment solutions do not fall into the hands of others.

1.8 Comments Regarding the Instructor's Assignment Solutions

As a matter of policy, only the solutions for problems that require written-English answers will be posted. Solutions to the problems for which code must be written will not be posted. There are two main reasons for this. First, there is typically no one correct solution to a programming problem and the instructor does not want to advocate one particular correct solution over all others by posting his solution. Second, the instructor would like to eliminate the possibility that students, in future offerings of the course, might plagiarize from his solutions. **Students are quite welcome to request to meet with the instructor in order to view the instructor's solutions to programming problems as well as discuss these solutions with the instructor. The student will not be permitted to make a copy of the solutions, however.**

1.9 Git Repository for the C++ Lecture Slides

The C++ lecture slides have a companion Git repository that is hosted by GitHub. This repository contains numerous examples and exercises. The URL for the main web page associated with the repository is:

`https://github.com/mdadams/cppbook_companion`

The URL for the Git repository itself is:

`https://github.com/mdadams/cppbook_companion.git`

1.10 Programming Assignment Grading

The software that is submitted as part of a programming assignment is subjected to fairly close scrutiny during grading. Therefore, it is important for the student to impose a high level of quality control (e.g., through testing and code inspection) on any code that is submitted. Although the course is obviously trying to teach students how to program in C++, the course has another equally important objective: to help students develop the skill of being able to effectively test software in order to ensure that it works correctly and meets all required specifications. The importance of good testing skills cannot be overstated, as such skills are critically important to employers. Employers want to hire individuals who can write code that works correctly. An individual, who is unable to test their code effectively, will not be able to write correctly functioning code. Correctness of code can only be ensured through effective testing.

1.10.1 How Software is Evaluated

The exact manner in which the student software is evaluated will vary from one assignment problem to another, but the general approach taken will tend to be similar to that described below.

In the case of complete programs, the evaluation typically consists of the following. The program is built using CMake in conjunction with the CMakeLists file provided. The program is executed with various different input data and/or command line options (as appropriate) in order to ensure that the code behaves correctly. The particular test cases employed are very carefully chosen in order to detect as many bugs as possible. To whatever extent is possible the building and execution of the code (with various test cases) is done using automated scripts. In addition to the above tests, all of the source code undergoes a (manual) code inspection by the marker (i.e., the marker reads through the source code line by line looking for any problems). The code inspection greatly increases the amount of time required for marking, but is necessary in order to catch problems that cannot be caught by automated tools (e.g., certain types of bugs, bad coding style, etc.). In the case of code that is not a complete program, the evaluation most likely will consist only of a (manual) code inspection.

It is absolutely imperative the file names used in the submission match exactly those specified in the assignment definition. Similarly, it is critical that the CMakeLists file meets the specifications required in the assignment handout. If these requirements are violated, this will cause many problems with the above evaluation process (e.g., the code will not build or execute correctly).

In the case of some assignment problems, the source code for a program is partitioned such that one or more source files contain code that provides some well-defined functionality and another source file provides code to test this functionality. Often, in such situations, part of the evaluation process will involve replacing the student's source file containing test code with one from the instructor (where the instructor's test code is very carefully designed to catch as many bugs as possible). For a scenario like this, when the evaluation results are conveyed to the student, the term "original code" is typically used to refer to the code exactly as submitted by the student, while the term "modified code" is typically used to refer to the code with the student test code replaced by the instructor test code.

1.10.2 How Evaluation Results Are Conveyed to Student

Although each programming assignment is submitted in electronic form, the grading feedback may be provided to the student in either electronic or hardcopy form. This grading feedback typically includes the following:

1. A summary that provides key information about the submission (e.g., student information and a list of the files included in the submission) and states the main evaluation results (e.g., whether the configuring/building and testing of the code was successful or not).
2. The contents of any log files generated during the configuring/building of code for cases where the configure/build failed.
3. The contents of any log files generated during the testing of code for cases where one or more tests failed. Some test cases involve comparing the actual output produced by a program with certain expected output. In such cases, when the actual and expected output do not match, both are typically included in the log (along with the difference in UNIX diff format).
4. Listings of various files included in the assignment submission (e.g., source-code files, build files, and readme files) with comments indicating errors, shortcomings, or other issues.

In cases where a log file is extremely long, its listing may be truncated to save space/paper. For example, on occasion, the errors produced during the compiling of a single source file have been observed to exceed 50 pages.

1.10.3 Policy on Late and Incomplete Assignment Submissions

Assignment submissions received after the submission deadline will not be accepted and will receive a mark of zero. Incomplete submissions that pass the assignment precheck and are received prior to the submission deadline will be accepted but will be penalized in accordance with the level of incompleteness.

1.10.4 How to Avoid Losing Marks Unnecessarily

In order to avoid losing marks unnecessarily on programming assignments, it is strongly recommended that you do the following:

1. **Enable and take notice of compiler warnings**. The compiler knows more about the C++ language than any mere mortal will ever know. So, why not benefit from the compiler's supreme wisdom when it is saying that the code is wrong? For example, it is not uncommon for students to lose marks for forgetting to return a value from a function with a non-void return type. Such a problem, however, is easily avoided by paying heed to compiler warning messages. If warnings are enabled, any reasonable compiler implementation will generate a warning if a function with a non-void return type does not return a value.
2. **Test your software thoroughly**. Ensure that your test code calls every function at least once (including all constructors, operators, etc.). Also keep in mind that calling a function only once is sometimes not sufficient for a thorough test. In particular, if a function must handle a number of special cases, the function should be called to test each of these cases. For example, if you were writing a sinc function, you should probably test it for at least two different values, one being zero and one being nonzero, since these cases are handled differently. In the case of template code, it is **extremely** important to ensure that every function/class is instantiated (i.e., used) at least once, since template code is not fully checked by the compiler until the template code is used.
3. **Before assignment submission time, always double-check that all assignment requirements are met.** After you think that you have a complete working solution to the assignment, do the following before you submit your work. For each problem statement in the handout, re-read the problem and, for each requirement stated in the problem statement, check that your software meets the requirement. Be particularly mindful of details like types for function parameters and return values. Be careful about const correctness.
4. Ensure that your Git repository contains the correct contents and ensure that the repository contents pass the **assignment precheck**.