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-interfaces (UIs) (e.g., command line or graphical); and
- application programming interfaces (APIs).

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, running, and testing) 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 will likely be very significant** (e.g., the penalty could easily result in a failing grade).

Except as explicitly required in the assignment specification, a program/library must not:

- write any output to standard output (e.g., `std::cout`) or standard error (e.g., `std::cerr` and `std::clog`)
- read any input from standard input (e.g., `std::cin`)
- create or access any files, directories, or other entities in the filesystem (e.g., sockets, devices, and so on).

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. A program must finish its task in a reasonable length of time for the inputs provided. Most programs should not take more than a few seconds to execute for the inputs given to them. If a program does not terminate in a reasonable length of time, this behavior will be assumed to be due to a bug (e.g., a bug that results in the code becoming trapped in an infinite loop). So, do not employ operations that artificially inflate the execution time of a program by significant amounts (e.g., delay loops or calls to `std::sleep`).

Some assignments require the student to write programs to test the functionality of some of the software that has been developed. Unless indicated otherwise, all such test programs must run without any command-line arguments being required and without any user input being provided.

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 1.6. 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 1.7. It is important that students pay heed to notifications from GitHub related to their Git repositories in GitHub Classroom, as such notifications may contain important information regarding course assignments. Students who disable such notifications or direct them to email addresses that are not checked regularly for messages do so at their own risk.

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

in Section 1.3. The Git repository must also capture the full revision history of the software being developed, as explained in Section 1.4.

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 1.11.3.

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 1.5.

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 in a file called `README.pdf` in PDF format. 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 `term` followed by the date (i.e., year and month) in which the term starts enclosed in double quotes, where the date is formatted as as a four digit year followed by a "–" character followed by the two digit month number; for example, if the term starts in May 2020, the date would be formatted as "`2020-05`";
5. the keyword `course` followed by a space and then the course name spelled in uppercase without any spaces and enclosed in double quotes; for example, the course name spelled in uppercase without any spaces would be `SENG475` for SENG 475 and `ECE596C` for ECE 596C;
6. the keyword `section` followed by a space and then the student's tutorial section (e.g., T01 or T02) enclosed in double quotes; if the tutorial section has an identifier that starts with "B" instead of "T", replace the "B" with a "T" when specifying the tutorial section; and
7. 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 of SENG 475 in the Summer (i.e., May–August) 2020 term 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"
term "2020-05"
course "SENG475"
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, 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 Git Repository Commit History

The revision history of a Git repository must capture all of the work on the assignment from beginning to end. Throughout the entire period of work on an assignment, the student is **required to make regular commits** to their assignment repository on GitHub. Moreover, each commit **must include a descriptive commit message** that accurately documents the changes made in the commit. Not including a detailed revision history may negatively impact the grade received on the assignment and possibly raise suspicions of plagiarism. Furthermore, not including any revision history at all **may cause the assignment submission to be rejected outright** (and given a mark of zero) due to a high likelihood of being plagiarized. Aside from teaching students good software-development practices, the requirement that a well-documented revision history must be provided also helps to protect the student by creating a written record of the student's work in the event that the marker might raise concerns of plagiarism.

1.5 The `assignment_precheck` Command

To eliminate the possibility of many basic mistakes in the assignment submission process, a program called `assignment_precheck` is provided that performs basic sanity checking on the assignment submission in a Git repository. For information on how to use this program, invoke it with no command-line parameters or options as follows:

```
assignment_precheck
```

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. Note that *url* can use either the HTTPS or SSH protocol. For illustrative purposes, we consider a few specific examples of using the `assignment_precheck` command in what follows.

Example 1. Suppose that we want to check the files in the most recent commit of the `master` branch of the repository with the URLs `https://github.com/uvic-seng475/repo.git` (for HTTPS) and `ssh://git@github.com/uvic-seng475/repo.git` (for SSH). We can accomplish this using the HTTPS protocol with the command:

```
assignment_precheck https://github.com/uvic-seng475/repo.git
```

Alternatively, the SSH protocol can be used with the command:

```
assignment_precheck ssh://git@github.com/uvic-seng475/repo.git
```

Example 2. Suppose that we want to check the files in the most recent commit of the `experimental` branch of the repository with URLs `https://github.com/uvic-seng475/repo.git` (for HTTPS) and `ssh://git@github.com/uvic-seng475/repo.git` (for SSH). We can accomplish this using the HTTPS protocol with the command:

```
assignment_precheck -b experimental https://github.com/uvic-seng475/repo.git
```

Alternatively, the SSH protocol can be used with the command:

```
assignment_precheck -b experimental ssh://git@github.com/uvic-seng475/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 and may possibly perform some additional checking. If the configure or build operation or additional checking fails for an assignment submission, the submission will still be accepted for marking but the submission is at **extremely high risk of receiving a failing grade** since the code is likely to be completely untestable in such circumstances. Consequently, it is advisable to ensure that the configure and build operations as well as any additional checking operations performed are all successful.

1.6 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. Since notifications are sent to this address, it is important that the email address provided be one that you check regularly for messages.

1.7 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.

1.8 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.

1.9 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.

Although no solutions will be posted for the assignment problems for which code must be written, the instructor has prepared solutions for these problems as part of his preparations for teaching the course. If a student would like to know how the instructor handled some aspect of a particular programming problem, the student is most welcome to meet with the instructor in order to discuss the instructor's solution. The instructor, however, will not provide a copy of his code to the student.

1.10 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

1.11 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.

1.11.1 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.

The evaluation process typically consists of the following. Any programs and libraries are built using CMake in conjunction with the CMakeLists file provided. Each program is executed with various different input data and/or command line options (as appropriate) in order to ensure that the code behaves correctly. Each library is tested using one or more test programs. 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 typically 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 some assignment problems, the source code is partitioned into two parts: 1) a library that provides some well-defined functionality; and 2) one or more application programs that serve to test the library. In such situations, part of the evaluation process will typically involve replacing the files containing the student's test code with files containing the instructor's test code (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.

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).

1.11.2 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 in length.

1.11.3 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.

1.11.4 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. **Use code sanitizers.** Code sanitizers are extremely helpful for finding bugs and can often detect very serious problems in code that appears to be (but is not actually) working correctly.
3. **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 (of nontrivial complexity) only once is rarely sufficient for a thorough test. 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.
4. **Use a code coverage tool**. It is recommended that a code coverage tool be used to assess the coverage achieved by the testing performed. For the software developed in this course, 100% statement coverage (i.e., every line of code is executed at least once) is strongly recommended (except possibly for code in some exception-handling paths).
5. **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 extremely careful about const correctness, as making mistakes related to const correctness will almost always result in code that will not compile under certain circumstances.
6. Ensure that your Git repository contains the correct contents and ensure that the repository contents pass the **assignment precheck**.