Project

1 Purpose

The primary purpose of the course project is to provide the student with the opportunity to use the C++ programming language in order to solve a particular problem of interest relevant to engineering.

2 Overview

Each student is to complete a project in which they develop software to solve some particular problem of relevance to engineering. The C++ programming language must be used. As for the particular problem to be addressed by the project, considerable flexibility is given in this regard. For students in ECE 596C, the project must be done individually (i.e., not in a group). For students in SENG 475, the project may be done individually or in a group with at most four members.

In the case of a group project, the size of the group is taken into consideration when the project is graded. A group with n students (where $n > 1$) is expected to produce a project that requires approximately n times the effort of a project produced by an individual student. Therefore, if an identical project were produced by both a group and an individual, the individual would receive a higher grade. In the case of a group project, each student in the group receives an identical grade.

The project has two components: 1) a proposal of the software to be developed, and 2) the software itself. Since the project accounts for a significant percentage of the final course mark, the student is expected to spend a fair amount of time and effort working on the project. The work on the project is undertaken in the following three steps (in order):

1. The student meets with the instructor in order to informally present the idea for the project and obtain permission to proceed with the preparation of a formal (written) proposal.
2. The student prepares and submits a formal (written) proposal that describes the project in detail in order to obtain permission to proceed with the development of the project software.
3. The student develops and submits the project software (which includes a brief video presentation).

A written proposal that was not approved during a preliminary project discussion will be rejected outright and not graded. Any submission of project software that does not correspond to an approved (written) proposal will automatically receive a grade of zero.

Students must allow for the fact that the instructor has **limited availability** for meetings. Furthermore, students must allow for the fact that a proposal takes a **significant amount of time to review**. It would, therefore, not be unreasonable to have to wait a week for a meeting with the instructor or for the instructor to complete the review of a (written) proposal.

3 Preliminary Project Discussion

Each student (or group in the case of a group project) is **required** to meet with the instructor in order to discuss the planned project with the instructor **before** starting to write the project proposal. By discussing the planned project in this way, the chance of the proposal later being rejected can be minimized. For example, a proposal might be rejected due to:

1. another student in the class already having chosen a similar project;
2. being too difficult to complete in the timeframe available for the project; or
3. not being of sufficient relevance to the course.

4 Proposal

The proposal is to provide a detailed description of the work that the student plans to undertake for the project. The proposal is intended to serve the following purposes:

1. To force the student to conduct a preliminary investigation in order to ensure that the project being considered is feasible to undertake, subject to the time (and other) constraints imposed by the course.
2. To compel the student to identify all of the methods required for the project as well as any other libraries or software tools that will be needed.
3. To ensure that the student has a clear plan as to exactly how they will undertake the work associated with the project (e.g., decompose the project into work items with a detailed work schedule).
4. To serve as a written contract clearly describing the software that the student promises to deliver.
5. To provide a detailed description as to how the software is to be run (i.e., a complete user's manual for the software).
6. To provide the instructor with an opportunity to provide feedback to the student regarding the scope and suitability of the proposed project, before work on the project begins.
7. To give the student an opportunity to practice their written communication skills.

The proposal should:

1. State the title of the project.
2. Identify the problem being addressed by the project, providing references as appropriate.
3. Specify all methods that will be used to solve the problem, providing references as appropriate.
4. Provide a complete description of the planned user interface for the software. This description must include sufficient detail that someone would know **exactly** how to run the software, including (but not limited to) information such as: a full description of any command line interfaces (e.g., command line parameters), a full description of any graphical user interfaces, and a complete specification of any file/data formats used.
5. Specify any restrictions/limitations imposed by the software (e.g., restrictions on input data).
6. Identify any libraries (other than the C++/C standard library) that will be employed (e.g., Boost, OpenGL, GLFW, GLUT, GLM, CGAL), being reasonably specific as to what functionality is required from each library.
7. Clearly distinguish between code that the student will develop entirely on their own and code written by others (such as libraries) that the student will use.
8. Include a carefully-planned schedule for the completion of the various stages of the proposed work.

The proposal is a **formal technical document**. So, all of the good practices for technical writing should be followed in the proposal (for example: use proper English and avoid overly colloquial language; strive for clarity, conciseness, and precision; use references to support statements made; and so on). Every page of the document must be numbered (with the possible exception of a title page), and key structural elements of the document, such as sections, tables, and figures, must also be numbered. Any equations that need to be referenced at later places in the document should be numbered and referenced by number. The first page of the proposal must include:

- the title of the project; and
- the full name, student ID, and **email address** of each student in the team working on the project.

The proposal should be **double spaced** and use a font size of **at least 10-point**. An appropriate length for the proposal will vary from project to project, as some projects will require longer descriptions than others. The goal should be to keep the proposal as short as possible, subject to the constraint that all of the required information must be provided. In other words, the length of the proposal must be reasonable for whatever project is being proposed.

It is critically important that the schedule for the proposed work be realistic. Otherwise, it is impossible to be confident that the project can be completed on time. This implies that some significant preliminary investigation is required **before starting to write the proposal** in order to determine what methods are needed, what libraries are needed, what steps are needed to complete the project (without overlooking anything that might take a significant amount of time) and an accurate estimate how much time each step will require.

A project proposal should be well organized, well written, clear and concise. Quality of writing is important. A poorly written or difficult to understand proposal is likely to be penalized significantly during grading. The proposal should present **clear evidence that the student has done sufficient preliminary investigation** to be confident that the project is feasible to complete in the time available. This implies that any relevant methods and any necessary libraries (or other software tools) should be discussed in sufficient detail to clearly demonstrate that the student has a good appreciation of exactly what work is needed in the project. Since a proposal requires a significant amount of preliminary investigation, **a high-quality proposal cannot be prepared in only a few days**.

4.1 Submission of Proposal

The proposal is to be submitted via e-mail in PDF format. (Any other format will be rejected by the instructor.) It is **very strongly recommended** that the student submit their project proposal for approval **as early as possible** in order to maximize the amount of time available to work on the project software.

4.2 Approval of Proposal

The instructor will review the submitted proposal and make a decision on whether to approve it. This decision is then conveyed to the student along with any comments or concerns about the proposal. If the proposal is not approved, the student is required to submit a revised proposal that addresses the concerns raised by the instructor. If the instructor's concerns cannot be addressed, an entirely new proposal would need to be prepared. The grade for the proposal is always assigned for the initial submission, even if one or more revisions of the proposal are required. The student is not permitted to start work on the project until the proposal has been approved by the instructor.

4.3 Changes to Proposal Subsequent to Approval

After the project proposal is approved, any change to the project definition requires the written permission of the instructor.

5 Software (Including Video Presentation)

The software developed for the project must be written in the C++ programming language, and **must be capable of being built and run on the machines in the lab used for the teaching of the course**. Note that only libraries that are available to all users on these machines can be employed. This is necessary in order to ensure that the instructor can build and run the software. The software must build with the software development environment used in the teaching of the course.

5.1 Video Presentation

The project software **must be supplemented by a short video presentation** about the software (which must have an audio track). The presentation should briefly introduce the software (using some slides) and provide a short demonstration of its use. The duration of the presentation **must not exceed 5 minutes**. A URL that can be used to view the video presentation is to be submitted along with the project software. The actual video and audio data for the presentation itself, however, are not to be submitted directly. Unless special arrangements are made in advance, **YouTube must be used to host the video for the presentation** (i.e., `http://www.youtube.com`). YouTube is free to use for this purpose and is a very reliable website for video streaming. (Do not use other mechanisms to provide access to the presentation video, such as Google Drive.) The video can be posted on YouTube as either a public or unlisted video, but it **must not be made private**; otherwise, the instructor will not be able to view it. Aside from being used to evaluate the project software, the video presentation will also be used to show to the public (including future students) some of the projects undertaken in the course. For advice on desktop capture software (for making the presentation), refer to Section 10.

5.2 Submission of Project Software

The mechanism for submitting the project software is similar to that used for programming assignments in the course. That is, the software is submitted via a Git repository obtained via GitHub Classroom. An assignment invitation URL will be sent to each student (or group). This URL can be used to gain access to a repository to be used for submission of the project software. For more details on the submission process for programming assignments, please refer to the programming-assignment handout. **When the repository containing the project software is ready for submission, the student must notify the course instructor of this by email.** The project software will not be deemed received until after the instructor receives such an email from the student.

5.3 Repository Organization

The Git repository used to submit the project software has some constraints on its organization. The top-level directory in the Git repository **must** contain the following:

1. a file `IDENTIFICATION.txt` that provides information about the project submitter as well as the project itself. This file is to follow the same format as the `IDENTIFICATION.txt` file used in the submission of programming assignments in the course. The assignment ID specified in this file should be `cpp_project`.
2. a file `README.txt` that should contain any additional information that might be helpful for building, installing, and running the software. This file must be in plaintext (i.e., ASCII) format.
3. a CMakeLists file `CMakeLists.txt` that facilitates the building and installation of the software via CMake. Note that this CMakeLists file must support the installation of the software, not just the building of it.
4. a file `demo` that is a script containing an appropriate sequence of commands to run a demonstration of the software. During installation, this script should be placed in the `${CMAKE_INSTALL_PREFIX}/bin` directory (where `${CMAKE_INSTALL_PREFIX}` denotes the value of the `CMAKE_INSTALL_PREFIX` variable of CMake). Any one of the following scripting languages may be used: Bourne Shell, Bash, and Python. The script must be compatible with the version of the applicable interpreter program (e.g., `sh`, `bash`, and `python`) installed on the lab machines.
5. a file `presentation.url` that contains a URL to be used to view the video presentation for the software. This file must contain only the URL on a single line. No other characters should appear in the file.

An example of a Git repository for a project can be found at the following URL:

`https://github.com/mdadams/uvic_elec586_project_example.git`

It is strongly recommended that students review this example, as this is likely to prove quite helpful. The student should run the `assignment_precheck` command on their repository before submitting it, as a basic sanity check. (The `assignment_precheck` command is described in the programming-assignment handout.)

5.4 Building and Installation Requirements

When the project software is built for evaluation by the course instructor using CMake, an out-of-source build will always be used (since in-source builds are considered a bad practice). Consequently, it is important that students use out-of-source builds when testing the build/installation process. The `CMAKE_INSTALL_PREFIX` variable will be used to specify an installation directory for the software. Since, for evaluation purposes, the software will be run from the installation directory, it is important to ensure that the software works correctly when run in this manner.

Let `$TOP_DIR` denote the top-level directory of the project software. Let `$INSTALL_DIR` denote the directory into which the software is to be installed. To build and install the software, a sequence of commands like the following will be employed:

```
cd $TOP_DIR
cmake -H. -Btmp_cmake -DCMAKE_INSTALL_PREFIX=$INSTALL_DIR
cmake --build tmp_cmake --clean-first --target install
```

To run the demonstration script, a command like the following will be used:

```
$INSTALL_DIR/bin/demo
```

The installation process for the project software must not create or access any files/directories except those under the directory `$INSTALL_DIR`. The execution of the project software must not create, delete, or modify any files/directories except those under the directory `$INSTALL_DIR`. Significant mark penalties will be applied for violations of this rule, since such violations could corrupt files needed for other projects being graded.

5.5 Software Evaluation Criteria

The software developed for the project will be evaluated using such criteria as the following:

1. Difficulty and effort required. The level of difficulty of the project chosen and how much effort is required to complete the project (relative to the number of students working on the project).
2. Correctness. The software functions correctly (i.e., has no bugs).
3. Design quality. The extent to which the overall structure of the software was well planned.
4. Usability. The software is easy to use.

5. Style. Good coding style is employed. There are many aspects to good coding style as discussed in the course, such as: const correctness, using meaningful identifier names, using consistent naming conventions, avoiding unnecessary global variables, making appropriate use of libraries, avoiding unnecessary code duplication, and so on.
6. Efficiency. The code is reasonably efficient. (The code does not need to be highly optimized, but it should not be grossly inefficient.)
7. Code documentation. The code is fully commented, and the comments are concise and understandable.
8. User documentation. How to run the software is clearly explained. The format of any input or output files used by the program are clearly specified.
9. Creativity. The extent to which the project is innovative, either in the choice of problem to solve or in the manner in which the problem is solved.
10. Coolness factor. The extent to which the software makes people say "Wow, that's cool!".
11. Effective demonstration. The software builds properly and the demonstration script runs successfully. The demonstration video presents the software well.

The correctness of the code is **extremely important**. The student is expected to test their software thoroughly in order to ensure that it has no bugs. Situations involving bad user input, invalid user requests, and such must be handled gracefully. Moreover, it should not be assumed that the software will only be run using the demonstration script and/or datasets provided. For example, other datasets may be used. Code that does not work properly will be penalized.

5.6 Additional Remarks on Project Software

In order to minimize the amount of written documentation that the student must prepare for the project, no formal design documents are required for submission. This said, however, the student is **very strongly advised** to spend adequate time to plan the overall structure of the software (i.e., the design phase) before starting to write any code. In the long run, a good design will save significant time later in testing and debugging.

6 Assessment

The grade for the project will be determined as follows:

Weight	Component
25%	Proposal
75%	Software (including video presentation)

7 Late Policy for Project Proposal and Approval of Project Proposal

Project proposals that are submitted late will receive a grade of zero. Any project whose proposal has not been approved by the deadline for the approval (not submission) of the proposal will result in an automatic grade of zero for the project software.

8 Late Policy for Project Software

The submission deadline for the software is very strict. This is due to the fact that this deadline is quite late during the term. Late submissions will be heavily penalized. Should the software be submitted late, a percentage d will be deducted according to the formula $d = 2^{\lceil n-1 \rceil}10$, where n is the number of days by which the software is late and $\lceil x \rceil$ denotes the smallest integer not less than x (i.e., the ceiling function). For example, software that is submitted up to 24 hours late would be penalized 10%, with the penalty growing rapidly (i.e., exponentially) thereafter.

9 Important Dates

The deadlines for the project proposal and project software will be posted on the course website.

10 Desktop Capture

Although many software tools are available for performing desktop capture, the use of the FFmpeg software is highly recommended, as it has many features and supports most mainstream computing platforms. FFmpeg is a popular cross-platform software tool for recording, converting, and streaming audio and video. This software can be used to perform desktop capture (with audio), and supports numerous operating systems, including Linux, MacOS, and Microsoft Windows. The FFmpeg software can be obtained from the FFmpeg website, which has the following URL:

https://www.ffmpeg.org

Some additional information on how to perform desktop capture with the FFmpeg software can be found at the following URL:

https://trac.ffmpeg.org/wiki/Capture/Desktop