COMP 3004 – *SCAPES* Deliverable #1

Due: Thursday, November 12, 2019 at 4:00 PM (afternoon)

Collaboration: This deliverable must be completed by **registered teams**

Design Pattern Presentation

Your team will select the best design pattern, *other than Façade*, out of all team members' Assignment #3 submissions. Alternately, you may choose a new design pattern for the presentation, but it has to be a pattern selected from the Gang of Four textbook (Gamma et. al). Do **not** choose Façade.

Your presentation will describe, in your own words, what the selected design pattern does and what problems it is meant to solve in general (this must be based on the Gang of Four's description of the pattern). You will describe how your *SCAPES* system uses the design pattern, why the pattern is necessary to your design, and how it is used in your design.

Presentations will take place in class on November 12, 14, 19, 21, and 26. The projected date of each team's presentation will be decided a few days in advance, but the exact time will be random. All teams are expected to be in the classroom at the start of their presentation day. Each team will be given five (5) minutes to explain and justify their choice and usage of design pattern. Presentations that go significantly over time or under time will be penalized.

Each team will submit their presentation slides in *cuLearn* before the Deliverable #1 due date. Modifications to the presentation slides after the due date will be disallowed.

Feature Implementation

You will implement the following *SCAPES* features, as specified in the project description posted in *cuLearn*:

- Create a new program
- Save a program to a file
- Compile an existing program that contains the following SCAPL instructions only: **`dci`**, **`rdi`**, **`prt`**, **`cmp`**, **`jmr`**, **`jmp`**, **`end`**, **`#`** (for comments)

Design requirements:

- The output of the compilation step must use an internal format designed by your team, consisting of **objects** representing the SCAPL instructions and variables of the program, as well as the associations between these objects. Among other things, these associations **must** represent the dependencies between instructions and the variables or labels that they reference. For example, if one instruction declares a variable or label, and a different instruction uses that same variable or label, your design and your code must make the link between the two usages of the same variable or label. Your object design should closely match the object model found in the Assignment #2 solution diagram. It **must** use the inheritance hierarchy and the polymorphism indicated in the solution diagram.
- The compilation output, in the form of internal format objects, **must** be saved to persistent storage. In order to do so, you will need to convert the objects and their associations to a linear form. This conversion is called **serialization.** One possibility is for your team to convert your objects to XML or JSON, as the Qt framework may contain library classes that can assist with this task. Regardless of your choice of serialization techniques, the output must be human-readable.
- Submissions where the compilation output (the serialized objects) cannot be viewed in persistent storage, or where the serialized objects are not human-readable, will not be graded.
- The design pattern used for your presentation must be implemented in this deliverable, as must the polymorphism inherent in compiling the different kinds of SCAPL instructions.

Implementation requirements:

- The Linux Ubuntu platform, as provided in the official course virtual machine (VM), will be used as the test bed for evaluating your code.
- All source code must be written in C++, and it must be designed and implemented at the level of a 3rd year undergraduate Computer Science student, following standard UNIX programming conventions, and using advanced OO programming techniques such as polymorphism and design patterns.
- Only libraries already provided as part of the course VM may be used.
- The *SCAPES* user interface (UI) will preferably be graphical in nature. A console-based UI will be an acceptable alternative, but will not earn full marks for any of the coding deliverables. In general, user features should be easily navigable, either as menu items and/or pop-up menus. The look-and-feel of the *SCAPES* system should be professional and consistent with commercially available UIs.
- The *SCAPES* system will run on a single host, with all its data stored on that same host.
- Data storage, both in memory and in persistent storage, must be organized for ease of retrieval and efficient use of space. There should be no duplication of information anywhere.
- Persistent data may be stored in flat files, or any other mechanism already available in the VM provided, including the Qt SqlLite library.

Submission requirements:

- Delivery and deployment of your code must be *turnkey*. This means that the user must be able to install all the software for the project with one (1) command, build it with one (1) command, and then launch the project with one (1) command. Specific instructions for building and launching the project must be included in a README file. Projects that do not conform to these requirements will not be graded.
- Your submission must contain a minimum of two (2) complete SCAPL-language programs, already stored in persistent storage. One can be the sample program provided in the project description, and the other program can be basic but must contain at least one (1) loop. Both programs must have a useful purpose (not just a random set of instructions), each program must use **all** the SCAPL instructions implemented for this deliverable, and both programs must be fully documented using inline comments.

Grading

Grading breakdown:

- *Design pattern presentation:* *20%*
- *Feature implementation:* *80%*
  - Create program 10%
  - Save program 10%
  - Compile program 60%

Format

Presentation slides must be submitted as a **PDF document**.

Coding deliverables must be delivered as a single **`tar`** or **`zip`** file consisting of all source code, data files, and configuration scripts, as well as installation, build, and launch instructions in a readme file. Do **not** provide object files and project executables as part of your submission.