Assignment 1

Creating a Design Document

Author: Jimmy Tran, Brian Crites

This project must be done in a group of two

Introduction

As programmers it is tempting to immediately start coding after receiving the specifications for a project. This often leads to creating the basis of a project and then writing less than ideal patches to try and force it to produce the correct output rather than crafting an elegant solution to solve the problem at hand. Creating a good solution which is maintainable and extendable requires planning and review rather than ad-hoc development and patching. One tool software engineers use to help in the software planning process is Unified Modeling Language (UML) or Object Modeling Technique (OMT) which help visualize a project's architecture. In this assignment you will create a design document for the specifications given in Assignment 2, which will be your first coding assignment. This will include a simple OMT diagram to represent your components and their dependencies and serve as the primary form of documentation for your project. You can read more about creating OMT diagrams from this slide deck.

Assignment

Your assignment is to create a design document that you will use as a basis when developing your Assignment 2 submission. The purpose of this project is to give you time to think through the different elements that will be needed to fulfill the specifications, and create a plan of action rather than simply ad-hoc coding a solution and trying to patch problems as they arise. It is highly likely that you do not have experience with the type of work you will be doing for the Assignment and there will be some gaps in your knowledge about how certain tasks are going to be achieved. This is very realistic to how most of your real-world development projects will go, and is natural.

You should focus your efforts on breaking the specifications down into subproblems, looking at how those subproblems interact with each other, and determining what the best classes, structures, or interactions are for accomplish each task (and by extension, the entire project). Additionally, you should take this time to develop some prototype code to help you better understand how certain unfamiliar systems work so you can get a better understanding of how they might function with your proposed system. Like any good agile method, what you write in the design document is a starting point for your design, not a road map that you must rigidly follow.

Assignment Repository

Start by using the GitHub Classroom link to create a fork of the template assignment repository by creating a new group or join the group that your partner has already created. Do not join a random repository, make sure you have communicated with a specific partner that you will be joining their group and only join (or create) that specific group.

When creating your group name use good C++ variable naming conventions for naming your group. This means your group name should contain only letters, numbers, and underscores for separating words. GitHub will allow you to use non-regular characters in your group but does not always translate those characters correctly when downloading and can cause errors in the bulk downloading process. If you use non-regular characters in your team name and it causes an issue during the download process you will receive zero points for the assignment.

Before you start adding any documents or code through a command line interface, make sure that you have correctly setup the git client to attribute your commits to your account (you can get a refresher on how to do this in Lab 01). You will need to do this on **every machine** that you use command line git on for this project.

Design Document Specifications

You will write your documentation as the README.md file for your Assignment GitHub repository, as this is commonly where documentation about a project (especially in open source projects) lives. Your documentation should have the following sections and should use proper markdown for formatting headers, code segments, links, and images within your README.md. Each of the below bolded portions should be its own h1 header.

- **Project Information:** The project already has a “CS100 RShell” header. Under that you should have the quarter and year as well as both partners names and SIDs.
- **Introduction**: Start by giving a *brief* overview of what your program is able to accomplish (or what it will be able to accomplish after it is finished in this case) as well as any design patterns that are used and a brief overview of how the inputs are ingested, transformed into classes and structures, and processed.
- **Diagram**: Create an OMT diagram (or a small number of separate OMT diagrams depending on how complex) that shows the classes in your design and how they interact. You must use a drawing program to create your OMT diagram. OMT diagrams that are drawn by hand and scanned/photographed will not be allowed. Websites and online applications such as Lucidchart and Google Docs as well as programs such as GIMP and Inkscape are capable of creating these diagrams. You should add the images

to the `images/` folder in your assignment repository which you can then embed in your README.md using markdown.

- **Classes**: Create descriptions of each class and/or group of classes that you plan on developing. This can be as simple as a description of what each class accomplishes and how, or a pseudo code level class definition. A class group would be a group of classes that all inherit from a single base class (composite classes are an example) and are therefore all closely related. For class group give a description of the base class, as well as the differences between each child class. Make sure it is clear how these classes interact to perform the ingestion, transformations, and processing that is described at a high level in the introduction.

- **Prototypes/Research**: Since you are likely unfamiliar with how the functions `waitpid()`, `execvp()` and `fork()` function individually and together you should create a small prototype function to test how these functions can be used together to execute small commands in a separate thread. In addition to a prototype for the main system functions you should create a small prototype for parsing user input into the different elements necessary for completing assignment 2. The parser does not need to be bug free or fully complete (although it must compile) but should represent your investigation into determining which parsing method you want to use for your assignment and the basics of a full parser (I suggest counting the number of times different types of elements appear in a given input as the output of your prototype). You should include the code prototype you used to do your testing in the `prototype/` directory and describe your findings and how you plan on using it in your assignment in this section (note these prototypes do not need an associated CMake file to compile them). Additionally, you will likely have questions about how connectors act in different situations. You should perform some preliminary testing of these different situations against the normal shell and write put any notes about the results in this section.

- **Development and Testing Roadmap**: Using the design you have set out above, create an ordered list of development tasks that need to be fulfilled before the system can be completed. Note that this list will be relatively ordered, as some tasks can be completed in parallel. This list should include not only primary development tasks, that is building of classes and function, but also should include creating both unit and integration tests. For each item in your list, you should create a GitHub issue and assign it to the person who is planning on completing it. Note that these assignments will not be final as some tasks will be more difficult than expected and others will be easier and additional tasks will need to be completed as your design evolves. This will simply serve as a plan of action for how the work will be split between partners. Finally, you should use markdown to create a link between the development and testing roadmap task and the associated issue.

When you create your design document, do not think only about the specifications in current assignment but also think of ways that this type of program is likely to be extended and make

sure your design is well positioned for these changes. This is an important exercise in software construction, as successful projects usually have new functions added to them and old functions updated and modified.

Adding Images to the README

When you create your OMT diagram you can upload the image to an `images/` directory in your repository, and then use that uploaded image in your README. In order for your image to correctly show when using the markdown image syntax, you need to navigate to the image on GitHub, right click and “copy image address” and use that address in your markdown. If you have done this correctly, the url should end with `?raw=true`.

Submission

Commit any images you create and code prototypes to the proper directories and commit the updates to your README.md to your fork of the assignment repo from GitHub Classrooms.

You will also need to add an **annotated** `hw1` tag to the commit that you want to be graded. Annotated tags are described in lab 2 and in the git documentation. `git push` will not automatically push tags to your repository. Use `git push origin hw1` to update your repository to include the `hw1` tag. If you need to update your tag, you can remove the old tag using `git push --delete origin hw2` and push an updated one to your repo.

Your project must also contain a names.txt file in the root directory which contains the name, SID, and email of each of the partners in your group. It should have the following format.

```
Brian Crites, 860XXXXXX, bcrit001@ucr.edu
Andrew Lvovsky, 860XXXXXX, alvov001@ucr.edu
```

Grading

Your documents will be graded using the following breakdown

Points	Description
25	OMT Diagram
25	Class Descriptions
25	Prototype/Research
15	Roadmap & Issues

10	Introduction
100	**Total**