CS 103 Portfolio Lab/Project

Overview

The goal of this assignment is to provide you the opportunity to work through the design and implementation of a program of your choice, based on your interest or related field of study. In addition, we would like you to apply object-oriented programming principles including inheritance and polymorphism (which are the specific focus of the project), unless it is truly not applicable. (But with that in mind, we would like you to look for projects that could incorporate these principles) Along the way, you will/should use many of the topics we've discussed throughout the semester. .

In addition, we'd like you to have a program that you wrote from scratch to place on your resume and to add to your portfolio. **Please note that you may not post our CS 103 assignments publicly**, as they are copyrighted by USC. You may furnish your projects and HW to potential employers upon request, but should do so in a private folder (or directly). Thus, it would be great to have something you **CAN** post publicly. **That is this portfolio project!**

Assignment Structure

This assignment will be a group assignment in teams of **2 or 3**. **No** team of 4.

The assignment will last for the next 3 labs.

- **Friday, 11/7 - Week 1: Team Formation ("Find your Co-Founder(s)")**: Form teams (**within your specific lab section**) and brainstorm project ideas based on the selection criteria below.
- **Friday, 11/14 - Week 2: Designing your API**: Begin implementation by defining objects, interfaces (member functions) and data members (i.e. header files).
- **Friday, 11/21 - Week 3**: **Build Week:** Implement as much of your program as possible.
- **Friday, 11/28 - Week 4: Thanksgiving Week** - work independently (remotely) or with your team to make progress towards your final product
- **Friday, 12/5 - Week 5: Soft Launch**: Demonstrate your program to your TAs (and other students if you like).

Team Formation: Since you will be choosing your own project ideas, it would probably be best to work with those that are interested in similar things. With your TA, feel free to spend a few minutes thinking what you might want to do and then write those ideas (with your name) on the whiteboard. At that point you could join another group whose idea you like, look for similar ideas and merge them into one, or recruit others to join your group! You may form teams of 2-3 of your choice, but best practices show that teams work best when there is some but not a great difference in proficiency of the team members, there is individual responsibility (each person takes their responsibility seriously), and regular communication occurs. So with that in mind, you may not just want to choose who you are sitting next to, but that can also be just fine if both parties commit.

Problem Selection

The first step is to brainstorm with your teammate what you want your program to do. As you settle on an idea you can talk to your TA interactively, but we will also ask you to submit a brief description for your TA to review.

A key component of your project is to ensure it contains appropriate and sufficient objects that are amenable to hopefully using inheritance and polymorphism. Inheritance with polymorphism works best when there are:

- Sufficient objects in the application you are designing. You should have at least 3 objects/classes and each one should have at least 1 non-trivial function (i.e. getter/setter).
- Generally, **polymorphism** works best when there are objects that are different ***types/kinds/variations*** of some more generic type and where the behavior of each of these ***types/kinds/variations*** is different in some way (i.e. so as to use virtual functions).

We strongly encourage you to think of an application that interests you and may relate to your desired career goals. However, if you are struggling, some ideas of areas that would provide ample opportunities.

- Games [this might be the easiest if you are really struggling to think of ideas] - Games provide ample opportunities to create objects, many of which may lend themselves well to inheritance and polymorphism, such as different types of players or characters whose behavior differs like the Connect4 game with Human vs. AI input or the Blackjack player vs. dealer. Another good genre would be card collection games (e.g. Pokemon go or TCG).
- Consider an app that you use commonly - e-commerce can be great, music player/library, calendar, or social media (this may be less amenable to inheritance and polymorphism) but think carefully,
- Simulations where objects interact (often) based on randomized input or files containing event ordering/timing data and where you will measure specific metrics based on that data (e.g. student wait times to see a TA or CP, traffic on a highway, etc.)

Finding Inheritance Relationships and Polymorphic Behavior

Consider each of our projects thus far (which you cannot use as your Portfolio project, sorry 🙁):

- **Project 1** (Twentyone): We could have defined classes for `Card`, `Deck`, `Player` and `Dealer` (and maybe even `Hand`). Now consider where we could use inheritance and polymorphism. Where do we have different kinds or variations where we could use inheritance and polymorphism? Well Player and Dealer seem to be similar objects with different behavior for when they hit or stay. The `Player` can hit or stay simply based on their desire, whereas the `Dealer` must hit at 16 and stay at 17. We could consider making a `Participant` base class which `Player` and `Dealer` derive from. There could be a `virtual` function in the base class `virtual hitOrStay()` and then **different** implementations could be made in `Player` (to just do a `cin` to allow the human to choose `h` or `s`) and the `Dealer` (to compute the score of their hand and compare to 16 or 17). This would meet the criteria we gave for this Portfolio.

- **Project 2** (Connect4): Again, we could have made classes for the `Board`, `HumanPlayer`, `AIPlayer` and `RandomPlayer`. Fairly quickly you see that the last three classes mentioned are just different types of players. So we could factor out a base class: `Player` and then have all 3 of the above simply inherit from it. To use polymorphism, we could define a virtual function (among other non-virtual functions) `virtual void makeChoice(int& row, int& col)`. Each derived class could then make their own implementation. `HumanPlayer` would do what you did in `getHumanInput()` in the PR2. `AIPlayer` could do what you did in `getUserAIInput()`, and `RandomPlayer` could do what we gave you in `getRandomInput()` in PR2. This would meet the criteria we gave for this Portfolio.
- Project 3 (Maze): You already used objects in this project. It's a bit harder to see where we could use inheritance and polymorphism, but here is one idea. We defined one particular file format (with `S, F, #, .`) But you could imagine wanting to support other formats (just like there are many image formats: `.jpg`, `.png`, `.gif`, etc.), maybe someone will come up with a different format that uses different characters, numbers (`0`=space, `1`=wall, `2`=start, `3`=finish) or even an actual image (e.g. `.png`) of the maze with pixels. So what we could do is create a base class: `MazeReader` with a `virtual bool readMaze(const char* filename);` and then created derived classes tha inherit from it and provide their own implementation of `readMaze()` to read that particular format: `OriginalMazeReader` (i.e. our current format), `NumberMazeReader` (for the 0,1,2,3 option listed above), and even `PNGImageMazeReader` for mazes given as an image. Each could write their own `readMaze()` function prototyped as `virtual` in the base class so that their different behaviors could be used, swapped out, replaced, etc.

Hopefully, this gives you some ideas of how to find those inheritance relationships and polymorphic behaviors (`virtual` functions) in your own project ideas that we'll talk about below.

Week 1 (Today - 11/7)

- Brainstorm ideas for the project you want to write.
- Identify objects that exist and any inheritance relationships that exist (that meet the "is-a" relationship)
- Identify the general requirements of the program include:
    - What input will you take and how will it be provided (file, command line, keyboard/`cin`)?
    - What processing will you do on that data?
    - What outputs would you like to generate and to what destination (`cout` / a file)?
    - What assumptions will you make to simplify your application and make it doable. It should not be too simple, and we do NOT want to discourage you from really getting your hands dirty and trying to write a program that you can proudly show off to others and potentially employers. At the same time, you have 3 weeks of labs and limited outside work time. So plan accordingly.
- Check in with your TA for advice as needed.
- **Deliverable 1**: Team names/emails and list of project idea brainstorms submitted via this form.

Week 2 (11/14)

- Finalize the requirements of your project and list of objects from last week.
- Sketch the header files thinking about what data members you'll need and what functions each will need. You can use this sample header template to get started.
- Think through the interactions between objects. What functions would other objects want to call on this object, what are the arguments the functions need to perform that task (in combination with the internal data members), and what those functions would return.
- Consider how you'll start your data: vectors, deques, basic arrays, etc.
- Adjust your requirements as necessary. If it is clear that it is too ambitious, scale back the scope of your project. If it is too easy, what can you add to ensure it has enough objects and uses inheritance and polymorphism and is not just a trivial re-expression of something we have already done together in class?
- By the end of this lab, it should be reasonably clear how the program will operate, even if you have not written code yet.
- Avoid the temptation to start writing the implementations until you have fleshed out a reasonable interface and members of each class.
- Only if you have time should you begin to code the implementations.
- Check in with your TA for advice, as necessary.
- **Deliverable 2**: Copy/paste the current progress of your class declarations (header files) to this form. While it does not have to be complete, substantial progress should be evident.

Week 3 (11/21)

- Work on the implementation and try to finish a majority of the program
- Don't write all the features at once, but introduce 1 or 2 features at a time and test them with a main() program to ensure they work. For example, if you were writing a game, test out code to perform initialization, then one turn, then you can add in other players and test multiple turns, adding the final game win/loss logic, at the end. Or, if you are modeling an e-commerce application, you could model a single product and its purchase by a user and then add the ability to represent multiple objects, add different behaviors for pricing, display, or selection of those items later.
- Break your tasks down. Maybe each teammate can write a separate class? Or maybe you want to program together as a team: one person typing, the other giving ideas and verifying that what is written seems correct.
- Check in with your TA for advice, as necessary.
- **Deliverable 3**: Copy/paste the current implementations of your classes (and main code) to this form. At least 1 class should be complete and substantial progress should be evident.

During Week 2-5

- Work on the project with your teammates outside of lab. Students who build a project they feel passionate about and spend time working outside of lab sessions will get more out of this experience. Make something that you can be proud of!

Week 5 (12/5)

- Review exercises for the final
- Demonstrate your program to the TA.
- Complete the form for your TEAM experience. You will be giving feedback about your teammate(s) and their contributions.

Evaluation

Your team will be evaluated based on its effort each week to assign credit for your labs each week. But it needs to be effort that leads to the milestones we outline above being done in those weeks. If it is not, then you will not be given credit for that lab even though you attended. Again, implement basic features first and ensure they work before moving on to larger, more grandiose plans and features.

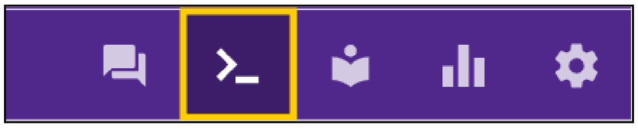
Do NOT Use AI

Practice writing the code yourself. Do NOT use AI. Use of AI will invalidate your lab credit.

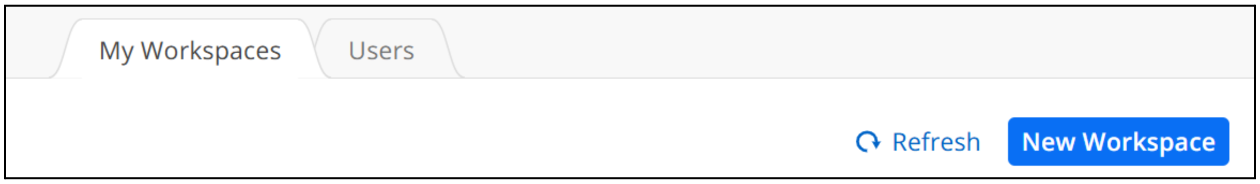
Group Coding Environment

Codio's group coding setup is poor, but **Edstem** has a nice feature called "**Workspaces**" where you can create a coding environment/project and then share it with other members in your group. **Note**: Have 1 teammate create the workspace using the directions below and then **share** it with the other team members.

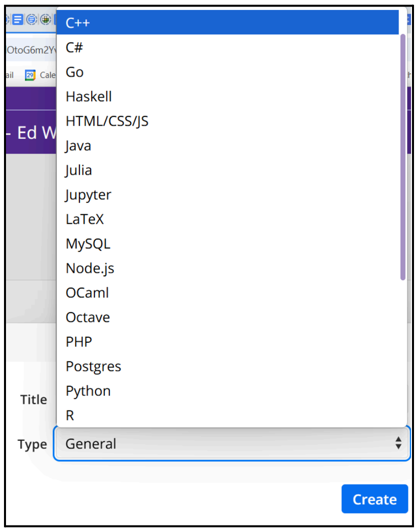
To start, login to EdStem. Then click on the Workspaces button on the top toolbar.



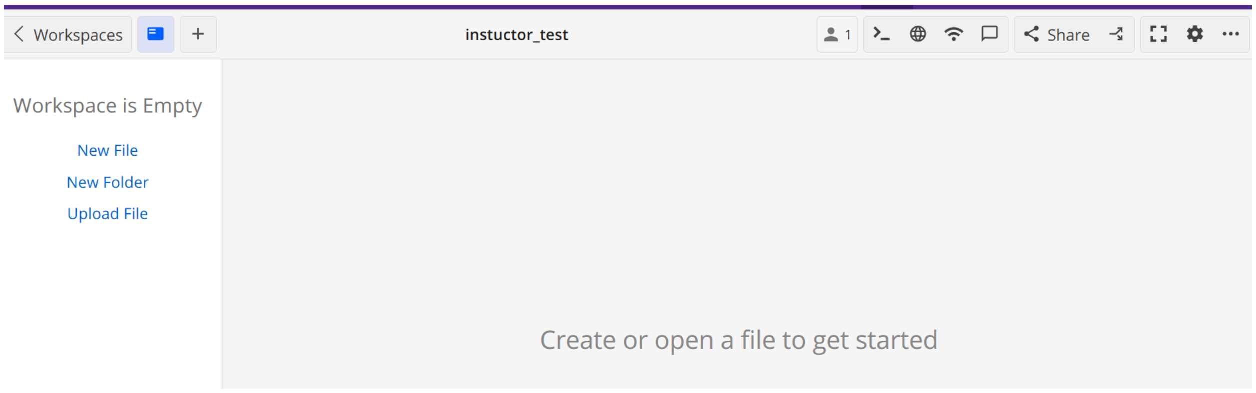
One member can then click on the `New Workspace` button:



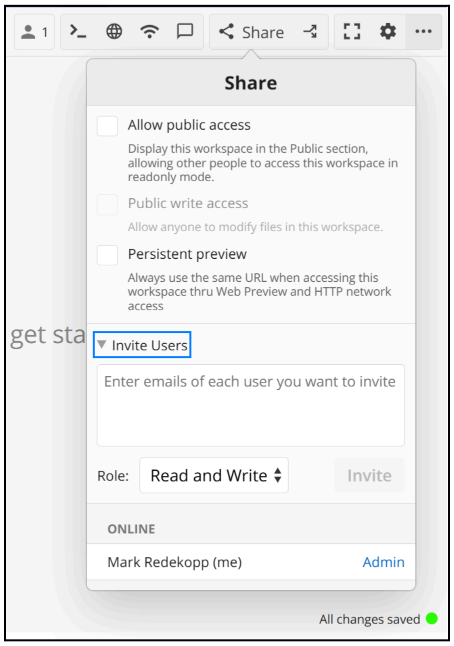
It will ask for a **Title** and **Type**. For **Title** use `Portfolio - <team name>`, where you substitute your own team name in. **Choose `C++` for the Type**.



You can then open your workspace and create new files, write code, open a **Terminal**, and compile and test your programs for yourself (again, there are no automated tests). **But most importantly,**



Then share it with your team so that other members can code at the same time in the same workspace. Simply click the `Share` button and enter the USC email addresses of each team member and ensure the role is `Read & Write`.



Share

Allow public access

Display this workspace in the Public section, allowing other people to access this workspace in readonly mode.

Public write access

Allow anyone to modify files in this workspace.

Persistent preview

Always use the same URL when accessing this workspace thru Web Preview and HTTP network access

Invite Users

Enter emails of each user you want to invite

Role: Read and Write

Invite

ONLINE

Mark Redekopp (me) Admin

All changes saved

Other team members should then see this workspace under their `My Workspaces` tab of the `Workspace` area in Edstem.