CSCI 103: Introduction to Programming

Lab 8
Objects: Classes and Structs

Lab Overview

- Goal: Practice writing objects
- Task 1:
  - You will be given a fully functional codebase for the pair-search game where you have to find matches in a grid of words
  - Currently it just uses functions as the primary abstraction
  - You will work with your TAs and in pairs to re-organize the code into classes and structs
- Task 2:
  - You will implement Word Ladder class and a UI to play the game.
- Checkoff: Manually check you off for active participation

Task 1: Pair Search

- Players take turns choosing 2 items from the grid, trying to find pairs (matches).
  - Make a match: get a point and go again
  - No match: no point and move to the next player's turn
- Game continues until all matches are made
- Player with the most points wins
- We will use strings (words) not pics

```
<---->   wolf  <---->  <---->  <---->
<---->  <---->  <---->  <---->  <---->
<---->   wolf   bird    bird   <---->
<---->  <---->  <---->  <---->  <---->
```

Task 1 Files

- 3 starter files on Codio
  - gameboard-using-functions.cpp/h: Completed code that breaks the code into functions (use as a reference).
  - gameboard.cpp/h: A copy of gameboard-using-functions.cpp that you will convert to use a class to model the game board.
  - main.cpp, main-using-functions.cpp: Driver code.

A Demo

- Compile the game

  `make pairs-functions`

- Run the game (`$ ./pairs-functions`)
- Input:
  - Dimensions of the grid (max 6x8). Try `3 4`
  - Enter a number of players. Try `2`
  - Enter players' names. Your choice.
  - At each turn, enter 4 indexes: row1 col1 for the first item to look at and row2 col2 for the second item to check. Try `0 0 0 1`
  - Keep playing
  - If you want to quit early, enter `-1`

Brief Overview of Data

Data:

- 2D array of strings (max `NROWxNCOL` but we only use `nr` by `nc`)
- 2D array of booleans (indicates which strings have been correctly guessed (again, only use `nr` by `nc`)
- 1D array of player names (`np` = number of players)
- 1D array of player scores (`np` = number of players)
-

Brief Overview of Functions

- `scramble` (shuffles/randomizes the strings board after we initially pick the words to use in initGameBoard)
- `initGameBoard` - initializes the 2D array of strings and guessed Booleans
- `printGameBoard` - Shows the board using `<----->` for unguessed words
- `checkMatch` - Takes the two sets of row/col indexes and checks if the words match and updates the board and guessed arrays. Returns true if a match was made.
- `isGameOver` - Determines if the game is over
- `main` - Primary game logic that coordinates appropriate function calls

Pair Work 1 (5-10 minutes)

Spend 10 minutes looking over the code in gameboard-using-functions.cpp and trying your best to understand it

- Try to answer each other's questions
- Write down/note any questions you cannot answer and ask them in the subsequent large group work

Large Group Review 1 (5-10 min)

Spend 5-10 minutes answering questions from the group

Introduce a Class

To find classes use…

- **Object Identification**: Identify objects directly described or related to the problem statement (Board, Player, etc.) [Look for the nouns in the description]
- **Object Invention**: Introduction of objects not directly specified from the problem statement but which facilitate decomposing the program into smaller pieces or grouping related data (e.g. a Location to pair together a row and column index)

Introduce a GameBoard class

- Should model the GameBoard (grid) of words and related operations
- As a large group write the class declaration
  - Identify what data would be best to make data members
  - Hint: What data is used across two or more operations related to the grid/gameboard or needs to live beyond the span of a single operation. Those should be data members
    - Arguments used in only a single operation should stay as an argument
  - Identify the member functions
    - Leave functions not directly pertinent to the GameBoard as global-scope functions

Pair Work 2 (10-15 min.)

Reformulate the relevant functions as member functions of the Class

Update main() removing data declarations that are now data members and updating function call locations

- If you don't finish, we will give more time after a brief explanation of the next part

Group Challenge (After You Finish Task 2)

- Challenge (please try if you have time)
  - Once you get the game working with the GameBoard class in gameboard.cpp, try adding a `Player` struct. Create a new header file for this struct.
  - The `Player` struct should hold the player name and score together in one.
  - Update the main.cpp code to use this Player struct.

Task 2: Word Ladder

- Word ladder is a single-player word play game. The player must connect the starting word to the target word by a sequence of words.
- Example:
  - Starting word: `SEAL`
  - Target word: `BALL`
  - Valid word ladder: `SEAL` -> `SELL` -> `BELL` -> `BALL`

Rules

- To jump from the current word to a new word, the new word must differ from the current word by exactly one character. The new word must also be a valid English dictionary word.
- If the player makes 10 valid jumps and does not reach the target word, the player loses the round. Invalid jump attempts (e.g. jibberish input words) do not count.

Overview of implementation

- Note: We are asking you to start **from scratch** on this task
  - You will get credit if you just give an honest effort
  - We don't expect you to finish in the giving time but can keep working on this as an extra practice exercise.
- start a new round (i.e. randomly select a starting and target word from the word bank)
- process a jump attempt from the user (i.e. read in a word from the user, determine if the constitutes a valid jump, make the jump if so)
- print the history of player jumps
- inform the user if they win (i.e. reach the target word) or exhaust their 10 valid jumps

Overview of Dictionary class usage

- You can include dictionary.h wherever suits you. You can create an instance of the dictionary class like so:

```
Dictionary yourdictobj("/usr/share/dict/words");
```

- You can check if a specific word is in the dictionary like so:

```
std::string word = "ocean";
if (yourdictobj.isWordValid(word)) {
  //
} else {
  //
}
```

- NOTE: Do not modify dictionary.cpp nor dictionary.h.

Pair Work (30 min) and Check-Off

- Continue working in pair to get your task 2 code working.
- Ask for help if you are stuck.
- Once 30 minutes remain, show your work to TA/CP. You'll be given credit if an honest attempt was made.
- Think about connections to your HW that used objects.