Programming Project 7 Instructions

100 Possible Points

Due: Sat Aug 13, 2022 12:00pm

Attempt 1 | IN PROGRESS Next Up: Submit Assignment | Add Comment

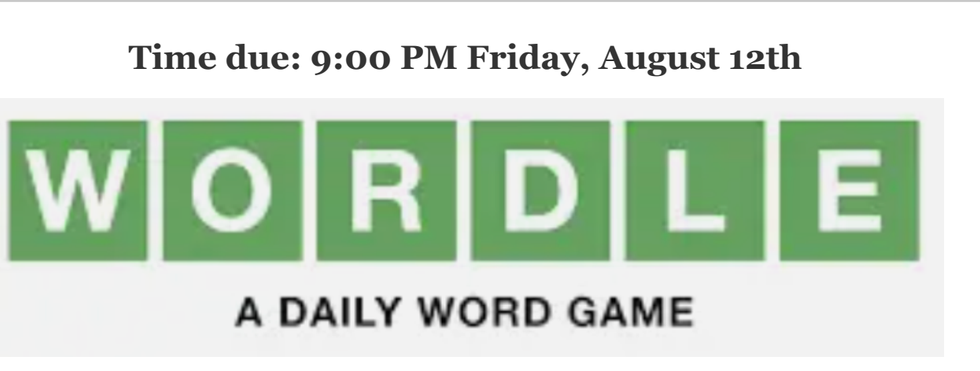
Unlimited Attempts Allowed
Available: Jun 21, 2022 11:00pm until Aug 13, 2022 10:00pm

Details

Instructions

Programming Assignment 7
Wordle!

Time due: 9:00 PM Friday, August 12th



Introduction

Wordle is a web-based word game created and developed by Welsh software engineer Josh Wardle, and owned and published by The New York Times Company since 2022. Players have six attempts to guess a five-letter word, with feedback given for each guess in the form of colored tiles indicating when letters match or occupy the correct position. Wordle generates a single daily puzzle, with all players attempting to guess the same word. The game gained a large amount of popularity in December 2021 after Wardle added the ability for players to copy their daily results as emoji squares, which were widely shared on the social media. If you are unfamiliar with Wordle, please review the rules of the game and you can also play for fun to get some practice with the game.

As the above description points out, I have made a few small changes from the one you can play online. First, the code you will be working with will allow for up to eight attempts to guess the word. Second, the code you will be working with supports a curated set of five-letter words so not every single five-letter word might be valid and playable with the code.

Your task

Your assignment is to complete this C++ program skeleton (XCode VS2022 VS2019) to produce a program that implements the described behavior. (We've indicated where you have work to do by comments containing the text TODO. Please remove those comments as you finish each thing you have to do.) The program skeleton you are to flesh out provides alot of code but you will be focusing your attention on the classes: Piece, Move, Score, Board and Wordle. You will need to complete code for the Move, Score, Board and Wordle classes. Details of the interface to all of these classes are in the program skeleton, but here are the essential responsibilities of each class:

In order to understand how the skeleton works, let's look at the LETTER enumeration and the class Piece. The LETTER enumeration is defined in the file Enums.h and is shown below:

<enumeration> LETTER
A B C D E F G H
I J K L M N O P Q
R S T U V W X Y Z
NOTVALID

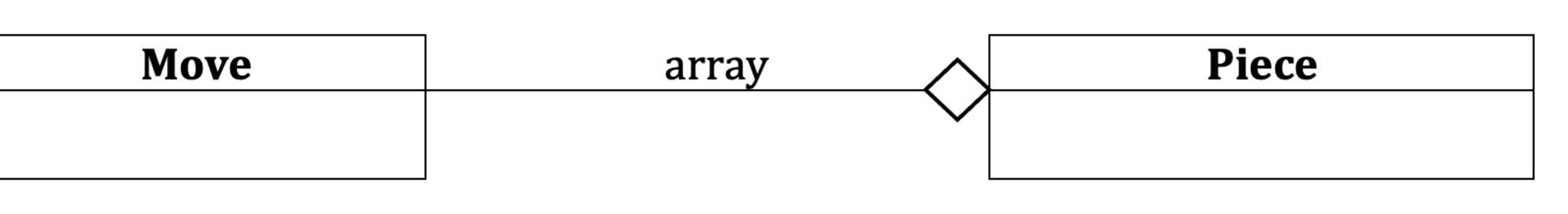
This enumeration defines a value for every english letter plus another value for any invalid other ASCII character value that might be entered (such as ! or @ or #). This enumeration is fully defined and needs no changes to be made by CS31 students.

The Piece class represents a single played letter in one round of play in the Wordle game. Please note the class diagram shown below:

Piece
- mLetter : LETTER
+ Piece( ) + Piece(c : char) + Piece(s : string) + getLetter( ) : LETTER + getLetterAsString( ) : string

The constructor of this Piece class enables client code to create a Piece from a single character or from the first letter of a string. By default, the Piece will have an NOTVALID LETTER value. Once defined, this class enables client code to access the underlying LETTER or convert that single LETTER value into a matching string representation. This class is fully defined and needs no changes to be made by CS31 students. If you scroll down, you will find driver code I have supplied in the assignment instructions that show you how you can work with objects of the Piece class.

A five-letter word played in one round of play in the Wordle game is represented by the class Move. A Move is made up of a set of five Pieces, as documented in the UML diagram shown below:



Please note the class diagram shown below:

Move
- array : Piece[REQUIREDLENGTH]
+ Move( ) **+ setPieces(s : string) : void** + setPiece(i : int, c : char) : void + setPiece(i : int, Piece p) : void + setPiece(i : int, s : string) : void + getPiece(i : int) : Piece + to_string( ) : string

Each Move holds an array of Pieces. The length of the array is controlled by the constant REQUIREDLENGTH which is defined in the file Settings.h. By default, each Piece will be NOTVALID. Client code uses the operation setPiece(...) to specify a single piece of a Move object. The parameter i should be an index value into the array of Pieces between 0 and REQUIREDLENGTH - 1. If the parameter i is out of bounds, setPiece will throw a std::logic_error back at the calling code, rather than offset outside the valid bounds of the array. Similarly, the operation getPiece(...) allows client code to access one individual Piece of this move. The parameter i should be an index value into the array of Pieces between 0 and REQUIREDLENGTH - 1. If the parameter i is out of bounds, getPiece will throw a std::logic_error back at the calling code, rather than offset outside the valid bounds of the array. The operation to_string( ) returns the five-letter word represented by this Move. NOTVALID Pieces will be returned as the space character. The operation setPieces(...) needs to be completed by CS31 students. This operation should accept a five-character string and set each Piece of this Move object to the character provided in the string parameter. If the string parameter is not five letters long, this operation should throw a std::logic_error back at the calling code. Please see the TODO comments in the class for further information. If you scroll down, you will find driver code I have supplied in the assignment instructions that show you how you can work with objects of the Move class.

Each round of play in the Wordle game needs to be scored, letter-by-letter, against the correct answer of the game. This scoring gets performed by the class Score which works with the enumeration ANSWER. The ANSWER enumeration is defined in the file Enums.h and is shown below:

<enumeration> ANSWER
RIGHT
WRONG
MAYBE

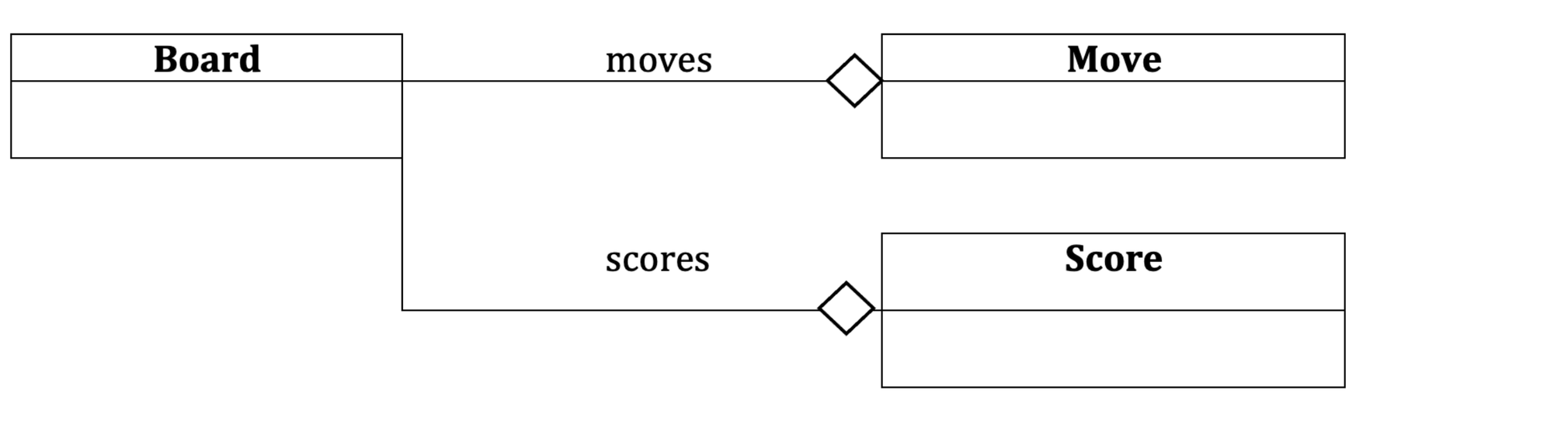
Once scored, each Piece in a Move will either be RIGHT, WRONG or MAYBE. RIGHT means that exact letter is in the same matching place when comparing the Score's Move to the correct answer. WRONG means that exact letter has no place in the correct answer when comparing the Score's Move to the correct answer. MAYBE means that exact letter is found in the correct answer but is located in the wrong position of the Score's Move.

Each Score holds an array of ANSWER. The length of the array is controlled by the constant REQUIREDLENGTH which is defined in the file Settings.h. Please note the class diagram shown below:

Score
- array : ANSWER[REQUIREDLENGTH]
+ Score( ) **+ Score(move : Move, answer : Move)** + getAnswer(i : int) : ANSWER **+ isExactMatch( ) : bool** + to_string( ) : string

By default, each ANSWER value will be WRONG. But if supplied two Moves, one for the played word and one for the correct answer, a Score constructor should properly define each ANSWER value accordingly. This second constructor needs to be completed by CS31 students. Please see the TODO comments in the class for further information. Client code uses the operation getAnswer(...) to access an individual ANSWER value for the index parameter i. The parameter i should be an index value into the array of ANSWER between 0 and REQUIREDLENGTH - 1. If the parameter i is out of bounds, getAnswer(...) will throw a std::logic_error back at the calling code, rather than offset outside the valid bounds of the array. A Score object can be converted into a string by calling to_string( ). RIGHT answers will be printed as an 'R', WRONG answers will be printed as a '_' and MAYBE answers will be printed as an 'M'. This operation isExactMatch( ) needs to be completed by CS31 students. It should return true when the ANSWER array holds all RIGHT values. Please see the TODO comments in the class for further information. If you scroll down, you will find driver code I have supplied in the assignment instructions that show you how you can work with objects of the Score class.

The class Board is repetitively called to print out the state of the Wordle game in a string table shown with the end of each round of play. Each Board is made up of a set of Moves and Scores, as documented in the UML diagram shown below:



The length of the arrays of Score and Move is controlled by the constant TOTALROUNDSALLOWED which is defined in the file Settings.h. Please note the class diagram shown below:

Board
- mRoundCount : int - scores : Score[TOTALROUNDSALLOWED] - moves : Move[TOTALROUNDSALLOWED]
+ Board( ) + endRound(m : Move, s : Score) : void + display( ) : string **+ getCurrentRound( ) : int** **+ getMoveForRound(round : int) : Move** **+ getScoreForRound(round : int) : Score**

Each Board is tracking the current round and provides a trivial accessor for that member value. A Move and Score gets supplied to the Board with calls to endRound(...) which should

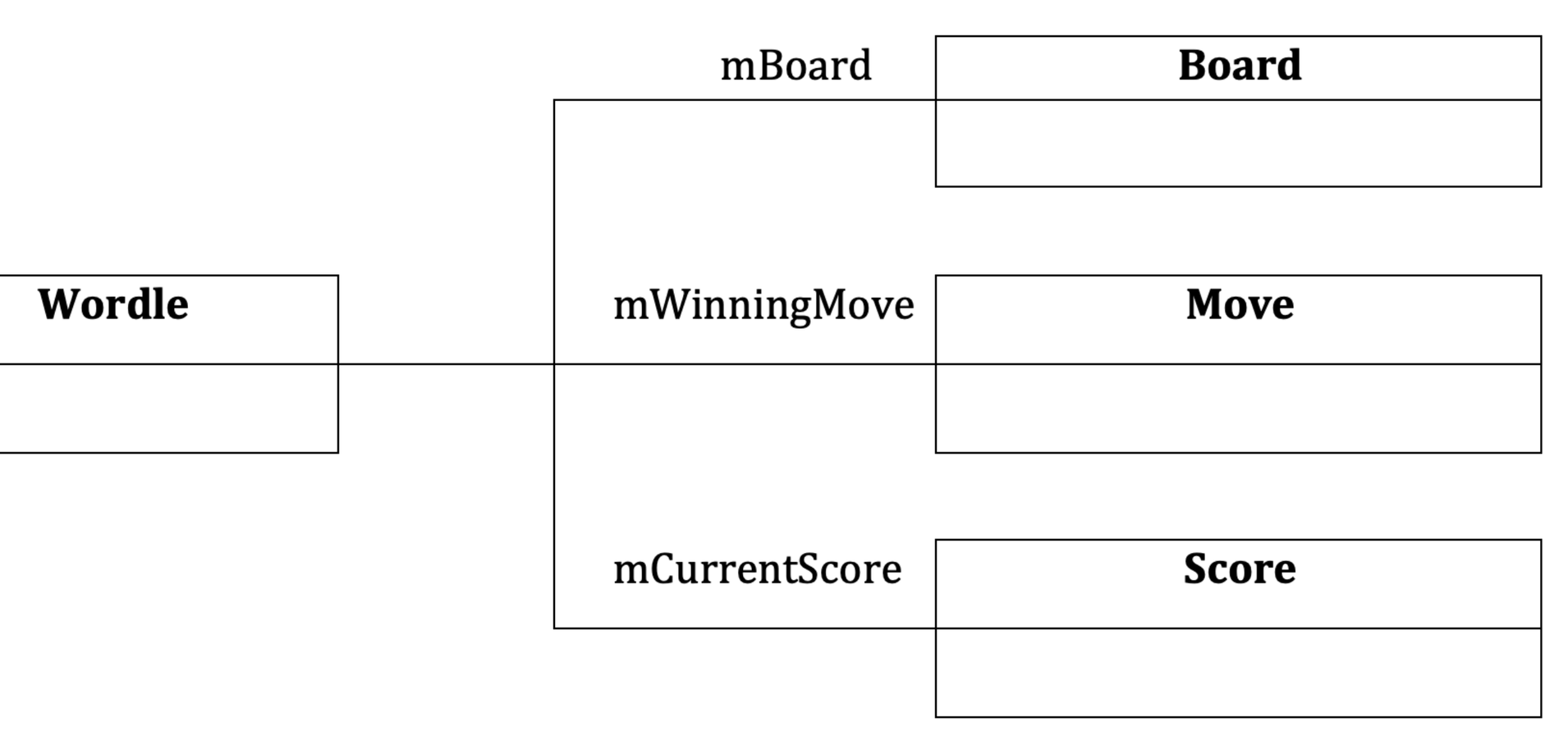
- copy the two parameters into array data members at the index of the current round and
- then increment the current round value.

Accessors are available to retrieve a Move or Score. In each case, the parameter round should be an index value into the array of Move or Score between 0 and TOTALROUNDSALLOWED - 1. If the parameter round is out of bounds, these two getters should throw a std::logic_error back at the calling code, rather than offset outside the valid bounds of the array. Please see the TODO comments in the class for further information. If you scroll down, you will find driver code I have supplied in the assignment instructions that show you how you can work with objects of the Board class.

The Wordle class plays the game with code found in main.cpp. At all times, the state of the game will be one of the possible outcomes defined in the enumeration GAMEOUTCOME. The GAMEOUTCOME enumeration is defined in the file Enums.h and is shown below:

<enumeration> GAMEOUTCOME
GAMEWON
GAMELOST
GAMENOTOVER

The Wordle class manages a Board, a winning Move with the correct answer as well as the current Move just recently played, as shown below:



The class Wordle is used in main.cpp to run the overall game. If you scroll down, you will find driver code I have supplied in the assignment instructions that show you how you can work with objects of the Wordle class. Please note the class diagram shown below:

Wordle
- mRound : int - mBoard : Board - mWinningMove : Move - mCurrentScore : Score
+ Wordle( ) + Wordle(m : Move) + Wordle(s : string) + display(msg : string) : string + isValid(turn : string) : bool + play(turn : string) : Move **+ endRound(m : Move) : Score** **+ determineGameOutcome( ) : GAMEOUTCOME** + gameOutcomeAsString( ) : string + gameIsOver( ) : bool + getBoard( ) : Board + answer( ) : string

~~The one operation that needs to be completed by~~ CS31 Students ~~is~~ need to complete determineGameOutcome( ). Based on the state of the Board, this operation should return the proper GAMEOUTCOME value. In addition, CS 31 Students need to complete endRound(...). Based on the Move parameter, this operation should create a Score object, increment the round counter and supply this Score to this game's Board, saving this Score as the current score value and returning it back to the calling code. Please see the TODO comments in the class for further information.

You are free to create additional public and private methods and data members as you see fit. However, the test cases will only be driving the public methods of the classes provided in the skeleton code originally.

The source files you turn in will be these classes and a main routine. You can have the main routine do whatever you want, because we will rename it to something harmless, never call it, and append our own main routine to your file. Our main routine will thoroughly test your functions. You'll probably want your main routine to do the same. If you wish, you may write functions in addition to those required here. We will not directly call any such additional functions. If you wish, your implementation of a function required here may call other functions required here.

The program you turn in must build successfully, and during execution, no method may read anything from cin. If you want to print things out for debugging purposes, write to cerr instead of cout. When we test your program, we will cause everything written to cerr to be discarded instead — we will never see that output, so you may leave those debugging output statements in your program if you wish.

Please read the posted FAQ for further assistance.

Additionally, I have created a testing tool called CodeBoard to help you check your code. CodeBoard enables you to be sure you are naming things correctly by running a small number of tests against your code. Passing CodeBoard tests is not sufficient testing so please do additional testing yourself. To access CodeBoard for Project 7, please click this link: https://codeboard.io/projects/161028 . Inside the files named Move.h, Move.cpp, Board.h and Board.cpp, Score.h and Score.cpp, and Wordle.h and Wordle.cpp, copy and paste the versions you have developed. CodeBoard uses its own main( ) to run tests against your code so you won't get any opportunity to provide a main( ) function. Click Compile and Run. However please be aware that no editing changes can be saved inside CodeBoard. In this anonymous user configuration, CodeBoard is read-only and does not allow for saving changes.

In an effort to assist CS 31 students with the contents of your .zip file archive, Howard has created a Zip File Checker which will echo back to you the contents of your .zip file. Please use this file checker to ensure you have named all your files correctly.

Programming Guidelines

Your program must not use any function templates from the algorithms portion of the Standard C++ library. If you don't know what the previous sentence is talking about, you have nothing to worry about. If you do know, and you violate this requirement, you will be required to take an oral examination to test your understanding of the concepts and architecture of the STL.

Your implementations must not use any global variables whose values may be changed during execution.

Your program must build successfully under both Visual C++ and either clang++ or g++.

The correctness of your program must not depend on undefined program behavior.

What you will turn in for this assignment is a zip file containing the following 22 files and nothing more:

- The files named **Piece.h, Piece.cpp, Dictionary.h, Dictionary.cpp, DictionarySegment.h, DictionarySegment.cpp, Enums.h, Random.h, Random.cpp, Settings.h, Singleton.h** and **Singleton.cpp** which are part of the supplied skeleton and do not need to be changed at all as well as the files named **Board.h** and **Board.cpp** that implement the Board class diagrammed above, the files named **Move.h** and **Move.cpp** that implement the Move class diagrammed above, the files named **Score.h** and **Score.cpp** that implement the Score class diagrammed above, the files named **Wordle.h** and **Wordle.cpp** that implement the Bunco class diagrammed above and the file named **main.cpp** which will hold your main program. Your source code should have helpful comments that explain any non-obvious code.
- A file named **report.doc** or **report.docx** (in Microsoft Word format) or **report.txt** (an ordinary text file) that contains in addition **your name** and your **UCLA Id Number**:
  A brief description of notable obstacles you overcame.
  A list of the test data that could be used to thoroughly test your functions, along with the reason for each test. You must note which test cases your program does not handle correctly. (This could happen if you didn't have time to write a complete solution, or if you ran out of time while still debugging a supposedly complete solution.) Notice that most of this portion of your report can be written just after you read the requirements in this specification, before you even start designing your program.
  How nice! Your report this time doesn't have to contain any design documentation.

To create this zip file, begin by creating a folder. The folder **must be** named with your **UCLA Id number**. Inside this folder, deposit the three desired files. Then zip this folder to create a single file. For example, if my UCLA Id number was 800012345, I would create a folder named 800012345 and then have a zip file to upload named 800012345.zip

As with Project 3 and 4 and 5, a nice way to test your functions is to use the assert facility from the standard library. As an example, here's a very incomplete set of tests for Project 7. Again, please build your solution incrementally. So I wouldn't run all these tests from the start because many of them will fail until you have all your code working. But I hope this gives you some ideas.... There are many others in CodeBoard for you to exercise your code on.

```
#include <iostream>
#include <string>
#include <cassert>
#include "Wordle.h"
#include "Score.h"
#include "Move.h"
#include "Board.h"
#include "Piece.h"

int main()
{
       using namespace std;
       using namespace cs31;

       // test code
       Piece p;
       assert( p.getLetter() == NOTVALID );
       p = Piece( "A" );
       assert( p.getLetter() == A );
       assert( p.getLetterAsString() == "A" );

       Move m;
       p = m.getPiece( 0 );
       assert( p.getLetter() == NOTVALID );
       m.setPieces( "hello" );
       p = m.getPiece( 2 );
       assert( p.getLetter() == L );
       m.setPiece( 2, 'z' );
       p = m.getPiece( 2 );
       assert( p.getLetter() == Z );

       Score s;
       assert( s.isExactMatch()  == false );
       assert( s.getAnswer( 2 ) == WRONG );
       m.setPieces( "hello" );
       Move theAnswer;
       theAnswer.setPieces( "stank" );
       s = Score( m, theAnswer );
       assert( s.isExactMatch() == false );
       assert( s.to_string() == "_____" );
       theAnswer.setPieces( "hello" );
       s = Score( m, theAnswer );
       assert( s.isExactMatch() == true );
       assert( s.to_string() == "RRRRR" );

       Board b;
       assert( b.getCurrentRound() == 0 );
       m.setPieces( "hello" );
       theAnswer.setPieces( "logic" );
       s = Score( m, theAnswer );
       b.endRound( m, s );
       assert( b.getCurrentRound() == 1 );
       assert( b.getMoveForRound( 0 ).to_string() == "HELLO" );
       assert( b.getScoreForRound( 0 ).to_string() == "__M_M" );

       Wordle game( "apple" );
       assert( game.answer() == "APPLE" );
       assert( game.gameIsOver() == false );
       m = game.play( "taboo" );
       s = game.endRound( m );
       assert( s.to_string() == "_M___" );

       cout << "all tests passed!" << endl;
       return( 0 );
   }
```

By August 12th, there will be links on the class webpage that will enable you to turn in your zip file electronically. Turn in the file by the due time above. Give yourself enough time to be sure you can turn something in, because we will not accept excuses like "My network connection at home was down, and I didn't have a way to copy my files and bring them to a SEASnet machine." There's a lot to be said for turning in a preliminary version of your program and report early (You can always overwrite it with a later submission). That way you have something submitted in case there's a problem later.

G31 Build Commands

```
g31 -c Board.cpp
g31 -c Dictionary.cpp
g31 -c DictionarySegment.cpp
g31 -c Move.cpp
g31 -c Piece.cpp
g31 -c Random.cpp
g31 -c Score.cpp
g31 -c Singleton.cpp
g31 -c Wordle.cpp
g31 -c main.cpp
g31 -o game main.o Board.o Dictionary.o DictionarySegment.o Move.o Piece.o Random.o Score.o Singleton.o Wordle.o
./game
```

Submission

Submit the following formats:

- .zip File

Please **DO NOT** use the Webcam, Files or Box or Google Apps buttons you see below.

Instead, use *Drag a file here* or click on *Choose a file to upload*.

Then click on *Submit Assignment* to upload the zip file.

Choose a submission type

Upload | More

Canvas Files or Drag a file here, or Choose a file to upload
File permitted: ZIP