Each exercise has a different difficulty level. Do <u>three</u> of the exercises – it is up to you to choose which ones you want to complete.

Once you're done, upload your source files on Desire2Learn, and also to your GitHub account. Create a repository called “CS 200 Projects” and upload your files here.

Table of Contents

Coordinates Class (Level 1)....................2
Dictionary (Level 2)....................3
Building (Level 3)....................4
Checkerboard (Level 4)....................5
Pioneers (Level 5)....................6

Coordinates Class (Level 1)

Class, private and public members, variables and functions

Write a class for a coordinate pair. Its private members should be the X and Y values of the coordinate pair, and it should have public Getters and Setters to access these values.

Additionally, there should be a function that can calculate the slope between two coordinate pairs, and return the slope as a fraction.

Have the user enter two sets of coordinate pairs in the program. Set up the CoordinatePair objects, calculate the slope, and output the result.

- CoordinatePair class
  - Member Variables:
    - x float private
    - y float private
  - Member Functions:
    - void SetX(float value) public
      Set the value of the private member variable x to the value that is passed in.
    - void SetY(float value) public
      Set the value of the private member variable x to the value that is passed in.
    - float GetX() public
      Return the value of the private member variable x.
    - float GetY()
      Return the value of the private member variable y.
    - float GetSlope(const CoordinatePair& other)
      Calculate the slope between the current object and the "other" object passed in.
      m = (y1-y2)/(x1-x2)

Dictionary (Level 2)

Class, public/private members, File I/O and Constructor

Write a simple dictionary application. You will use a class to represent one Word in the dictionary. The Word class will contain several strings: the word itself, an array of strings for each definition.

Your program should contain an array of Word objects. When the program is run, the dictionary will be initialized will words. In main(), you should create an output file, and output every word and its definitions to the text file. Use the GetEntry() function to get the Word's term and definitions to output to the text file.

- Word class
  - Member variables:

term	string	private
definition	string array[5]	private
defCount	int	private

  - Member functions:
    - Word constructor public
      Initialize defCount to 0.
    - void SetTerm(string value) public
      Set the value of term
    - void SetDefinition(int index, string value) public
      Set the value of the definition at the given index.
      Make sure to check whether the index passed in
      is a valid value first! Do not allow a segfault to occur!
    - string GetEntry() public
      Assemble all information – the term and all definitions
      into one string variable and return the string variable.
      Use \n and \t as special characters if you wish to have
      multiple lines.

Building (Level 3)

Class, public/private members, File I/O, 1D Array, Constructor, Sub-classes

You will write two classes for this exercise: **Building** and **Room.**

The Room object will contain information such as width and length of the room (private variables). The Room object should also contain Getters and Setters for width and length. Finally, it will also have a function GetArea, that will return the square footage of the room (width x length).

The Building object will will contain an array of Rooms (private). Make sure to also keep track of the array size and amount of objects in this array. You may need a constructor to initialize values.

The Building needs functions such as AddRoom, which will allow the user to specify the dimensions of this new room and put it in the array. (If the array is full, do not allow the addition of another room.)

Also add a function GetTotalArea, which will add the area of all of the building's rooms and return the total value.

Another function needed is GetAreaOfRoom, which will take an index of a room (do error checking!), and return the area of that specific room. (You can call the room's GetArea function, instead of calculating it manually!)

When the program starts, create a Building object. Ask the user to input all the room information, until either the Room array is full, or the user has chosen to not add any more.

Then, create an output file. Write out the total area. Beneath the total, you will display a list of all rooms and their individual areas.

Checkerboard (Level 4)

Class, public/private members, 2D array, Constructor, Sub-classes

Write a class that will represent a Checkerboard. Each Checkerboard space is a Tile, so you will also make a class to represent a Tile.

Tile object – It should store whether it is a dark or light space (you can use a boolean, char, string, etc.), as well as whether there is a piece on this board (you can use an integer to store if it has a player 1 or player 2 piece, or a char, or two booleans.)

Remember to make variables **private**, and have accessor functions like Getters and Setters. You might also create a GetSymbol or similar function that will return a character to represent the tile: For example, '#' for a dark space with nobody on it, ' ' for a light space with nobody on it, 'A' for player 1, and 'B' for player 2.

The **Checkerboard** object should contain a 2D array of Tiles. You can either set the board to be 8x8 or 10x10.

Make sure to use a Constructor to initialize the entire board – alternating dark and light spaces, as well as the checker starting positions.

You should also have a Draw function that will output the board using ASCII characters.

International draughts board, Michel32Nl at the English language Wikipedia, CC BY-SA 3.0

Drawing the board is good enough for this exercise. If you'd like, you can experiment further, such as adding a grid to the horizontal and vertical axes, and figure out how to allow users to move pieces – but this is for you to work on if it interests you.

Pioneers (Level 5)

Class, public/private members, 1D array, Sub-classes

Write a simulation of pioneers traveling along a 2000 mile trail. Along the way, the party will consume supplies.

Person object – A person should have a name, as well as a health counter (int, 0 – 100%), and a boolean for whether they're alive. Have member variables be private, and write appropriate accessor functions as needed.

Wagon object – A wagon has a speed (miles per day), keeps track of the Distance Traveled, Distance Remaining, the quantity of food, maximum food storage space, and maximum passengers. The Wagon should contain an array of Person objects.

Start the program by letting the user select a wagon from a few options. Wagons will vary by speed and storage space. Note that a *stagecoach* traveled at about 5 miles per hour. You can use this as a reference when calculating speed-per-day.

Once the user has selected, set up the new Wagon object. Allow the user to name their people, and initialize all people as alive and 100% health. The amount of people in the party will depend on the wagon's passenger count. The wagon should have its “distance traveled” variable set to 0, and “distance remaining” set to 2000.

Your program should continue looping until the Distance Remaining is less than or equal to 0, OR until all party members are “not alive”. Keep count of the amount of days that have passed. Each day, the wagon will move forward (adjust the distances), and every Person will consume some amount of the food. (Set the food consumption to either a random number or whatever you'd like)

Once the food reaches below some threshold (like 5 days of food remaining), trigger a “hunting” event, which will replenish the food count by a random amount. If a person is starving, remove some amount from their health counter.

During each day, there is a random chance for a person to get sick, which will knock some amount of health off their counter.

If a person's health reaches 0 or below, set their “alive” to false, and no longer have them consume food.