CSE220 – C++ OOP Options

Topics:

- Review Pillars of OOP
- Create a simple command line application
- Create and use inheritance & polymorphism
- Explore “IS A” and “HAS A”

Description:

This assignment will give you several options to complete. You only have to do one. Due to the timing in the term, any other options you do will be for your own benefit as there probably will not be time to grade other options for extra credit (negotiable).

If you do not choose a data-structures based option, I would highly recommend you complete one or more before CSE310.

Use the following Guidelines:

- Give identifiers semantic meaning and make them easy to read (examples numStudents, grossPay, etc).
- Keep identifiers to a reasonably short length.
- User upper case for constants. Use title case (first letter is upper case) for classes. Use lower case with uppercase word separators for all other identifiers (variables, methods, objects).
- Use tabs or spaces to indent code within blocks (code surrounded by braces). This includes classes, methods, and code associated with ifs, switches and loops. Be consistent with the number of spaces or tabs that you use to indent.
- Use white space to make your program more readable.

Important Note:

All submitted assignments must begin with the descriptive comment block. To avoid losing trivial points, make sure this comment header is included in every assignment you submit, and that it is updated accordingly from assignment to assignment.

Option 1 GrabBag Description:

One of the keys to winning card games is to understand probability. Card Counting is a much vaunted skill amongst Black Jack players and Casinos are always on the look out to prevent such a skill tipping the odds in favor of a player.

This assignment is going to create a very basic game to show how card counting works. We will make a Hi-Lo-Guess game. The player will be presented a card and guess if the next card will be higher or lower. The player will be given a point for each time they guess correctly.

Each round the game will calculate the probability of the next card being higher or lower and the probability of drawing any specific number.

This will be accomplished by simulating a deck of cards with a Grab-Bag data structure.

Grab Bag

The Grab-Bag is a simple data structure that can be created either with an array or a linked structure (We are going to opt for the linked structure). The Grab-Bag has two major functions:

- add things to the bag
- remove things **RANDOMLY** from the bag

Bag (65% specification)

You are to implement a Grab-Bag data structure in C/C++. You may choose to use either language as your code base.

You will create several structs/classes to accomplish this:

- Card
  - Suit
  - Value
  - Note: it is recommended you make Card a class so you can overload operators to make comparisons easier
- BagNode
  - Card* card
  - BagNode* next

You will maintain any and all necessary pointers to create and maintain the bag. This should be similar to a linked-list in approach. A head pointer and possibly a tail pointer are good ideas.

Your bag class will have several methods to facilitate playing the game:

- getCurrentSize():int – item count of bag
- isEmpty():bool/int – is the bag empty or not
- add(<item>) – Add the item to the bag
  - Hint – think about which add function you want to use from the linked list
  - Hint – Remember we remove randomly … so … do we need any fancy insertion?
- grab():Card – get a card out of the bag!
  - Remember this is random
  - Hint – arbitrary remove
  - Hint – don't forget your cases
- getFrequencyOf(<item>):int – get a count of how many of an item is in the bag
- empty() – empty the bag
- fillWithCards() – fill the bag with a new deck of 52!

I have not outlined perfectly every function that you may need. The above methods are REQUIRED though. You may implement any needed functions to help or perform other tasks. Also note that I don't have the parameters perfectly laid out for you. You will have to pass pointers around to make these functions work. Hint – look at the sample codes in the power-point.

Hi-Low-Guess Game

Your goal is to create a simple "Hi-Low-Guess" game for the player to play. The computer will act as the dealer and the player will be given 3 options: the next card will be higher, the next card will be lower, guess the next card.

If the player guesses higher or lower and they are correct they get a point. If they guess a specific card, they get 5 points!

If the card is equal value, the player neither wins or loses and is awarded 0 points and is prompted to guess again.

Each time the player is offered a guess they should be shown:

- All the cards that have been drawn so far
- The last card drawn that you're guessing against
- The probability the next card will be higher
- The probability the next card will be lower
- The probability of drawing a specific value

Other requirements:

- The player will keep guessing until they get their guess wrong
- You should re-populate the deck after the deck has been 2/3rds used
- Leave the last card drawn out though
- Make sure you tell the player that the deck was re-filled
- Count Aces as low (value 1)

Notes:

- The bag is the deck. Since you pick stuff out of the bag randomly, you don't have to shuffle it
- You should use H, D, S and C for Hearts, Diamonds, Spades and Clubs
- You should use numerical values 1 – 13 for card values
- If card is a C-style struct, make a printCard function that "pretty prints" your card for you
- While there are explicitly required functions/methods for the high-low game – YOU ARE EXPECTED TO WRITE APPROPRIATE FUNCTIONS/METHODS
- The player can play as long as they keep guessing correctly. Don't forget to refill the bag after 34 cards have been drawn. Don't forget to tell your user that you are doing so.

Extra Credit Opportunity:

+5 Make your Grab-Bag templated so it can store any Object that can be compatible with the functionality of your Grab-Bag. Create an option in your program to demonstrate the bag functioning with a different class or data type. This can simply fill the bag with 'stuff' and then empty it through grab() calls.

BIG GIANT NOTE: If you do this option, templates can't be shared across files. All your GrabBag code will go in the .h file.

Sample output:

```
Cards drawn:
[K-H] [4-S] [7-D]

Cards left in deck: 49
Probability of next card being higher: 46.94%
Probability of next card being lower: 46.94%%
Probability of next card being the same: 6.12%

Probability of next card being:
A     8.16%
2     8.16%
3     8.16%
4     6.12%
5     8.16%
6     8.16%
7     6.12%
8     8.16%
9     8.16%
10    8.16%
J     8.16%
Q     8.16%
K     6.12%

Last card: [7-D]
Points: 2
Choose option:
1 – Next card will be higher
2 – Next card will be lower
3 – Guess exact value
```

Use cases:

- If the user selects 1 or 2, draw the next card.
  - Award a point if the user guessed right and prompt them for their next guess showing all the above information.
  - If the user guesses wrong. It's game over. Tell them "GAME OVER" and show them their score. Exit the game.
  - If the card is the same value as the previous card, then tell the user "Same card" and give them 0 points and prompt them for their next guess showing all the above information.
- If the user uses option 3:
  - Prompt the user for the value they want to guess.
  - Draw the next card.
  - If the value matches, then give the user 5 points and prompt them for their next guess showing all the above information.
  - If the value doesn't match, it's game over. Tell them "GAME OVER" and show them their score. Exit the game.

What to turn in:

Create a zip file to submit:

- <lastname>_<firstname>_hw5.cpp
- <lastname>_GrabBag.h
- <lastname>_GrabBag.cpp
- <lastname>_Card.h
- <lastname>_Card.cpp
- Makefile

If you do the extra-credit, then there won't be a _GrabBag.cpp, everything will have to go into the .h file. Add any other files you need to prove your Template works.

Name your zip file <lastname>_<firstname>_hw5.zip

Submit to the course canvas.

Option 2 Basic Rectangle City Generator Specifications:

The software will generate a world using objects from an inheritance/polymorphism hierarchy. You will store these objects in a dynamic 2D array.

User Interface:

When the program starts you will ask the user to input a width and height for the world. This should be between 20 and 50. This will determine how big the world will be and how many objects will be created.

Object Orientation:

The software will consist of several classes:

- Zone
  - ← Children of Zone:
    - HighDenResidential
    - AverageDenResidential
    - LowDenResidential
    - Commercial
    - Industrial
    - RecreationSpace
- City class

Each terrain type will have rules on how it should be shown and what kind of things can be its neighbors.

Zone Class

- Abstract base class
- Properties
  - type:char
- Methods
  - toString(): std::string (#include <string> is allowed)
  - generateNeighbor:Zone
- Friend
  - Overload std::cout <<

City Class

- Properties:
  - width:int
  - height:int
  - cityData:Zone**
- Methods:
  - City(int width, int height) – constructor, takes in width and height then generates the city
  - -generateCity():void – private method generates the city using the outlined algorithm
  - +toString(): std::string – return a string of the city to print (see below sample output)
- Friend:
  - Overload std::cout <<

HighDenResidential

- 'H'
- Should generate a neighbor object with these probabilities:
    - 40% - HighDenResidential
    - 20% - AverageDenResidential
    - 0% - LowDenResidential
    - 20% - Commercial
    - 10% - Industrial
    - 10% - RecreationSpace

AverageDenResidential

- 'A'
- Should generate a neighbor object with these probabilities:
    - 20% - HighDenResidential
    - 30% - AverageDenResidential
    - 5% - LowDenResidential
    - 20% - Commercial
    - 5% - Industrial
    - 20% - RecreationSpace

LowDenResidential

- 'L'
- Should generate a neighbor object with these probabilities:
    - 0% - HighDenResidential
    - 20% - AverageDenResidential
    - 40% - LowDenResidential
    - 5% - Commercial
    - 0% - Industrial
    - 35% - RecreationSpace

Commercial

- 'C'
- Should generate a neighbor object with these probabilities:
    - 20% - HighDenResidential
    - 20% - AverageDenResidential
    - 5% - LowDenResidential
    - 30% - Commercial
    - 20% - Industrial
    - 5% - RecreationSpace

Industrial

- 'I'
- Should generate a neighbor object with these probabilities:
    - 20% - HighDenResidential
    - 5% - AverageDenResidential
    - 0% - LowDenResidential
    - 20% - Commercial
    - 50% - Industrial
    - 5% - RecreationSpace

RecreationSpace

- 'R'
- Should generate a neighbor object with these probabilities:
    - 5% - HighDenResidential
    - 10% - AverageDenResidential
    - 25% - LowDenResidential
    - 15% - Commercial
    - 0% - Industrial
    - 45% - RecreationSpace

Algorithm breakdown:

Setup:

1. Get size of array from user
2. Create Zone 2D array
3. Divide screen width/height by user inputted size
4. Create first piece of Zone in upper left corner
    a. Pick randomly
    b. Make sure you pass in the pixel width height to the constructor

First row:

1. Generate the second object by asking the upper left corner object to generate a new Zone object
2. Store the new Zone in the second index
3. Continue down the row by asking the object "to the left" to generate the new object

Subsequent rows:

1. The first item in the row will ask the previous row's first item to generate the object
2. Each item in the rest of the row will ask the object "up from" the current index and "to the left" of the current index to generate objects
3. Flip a coin and store between the two objects and store the winner in the array
4. Continue

UH WHAT?!

Take your time and think about it. It's not actually a hard algorithm. You'll be building a nested for loop to generate your Zone. You'll iterate the outer loop on the Y and the inner loop on the X

Then you build an if…else if…else chain to make sure you generate the Zone correctly.

Output

After the Zone is generated … draw it to the screen using cout

Here's a sample of one I generated 20x20:

```
HIRCAARCCHAARRLRRRRC
IARCAACCLAAACLCICHRC
IHHHAHIIRCHCCACICHLL
IHCACHCILIAALCHRCCCC
HAACHHAILLLLCCHHICII
CLLRAHACACCRCAAAHCIC
HIIHHHCHARLRACLAICHA
AIIRRRCCCALLLCRAIRRR
RLALARHIHILCLRRRRLRR
RLLRLRRIAIAIIHHLLCLL
RLLRRRCCRIIHIICLACCR
RALRRHARLLRRIIHHRCAA
LRRCRCACRRHACIHHRCAC
RRRLHHALRRCHCCHHHCAA
CRHIHICCRLIICAAIHHHC
CCHIHIIRRLLRRAARAHHA
CHAHCHCIRRRRLRCIRCRL
HHLRCACCRCAHRLHAAAAA
IHRCCCCCHICHHLLCCRCA
IIIHHAARRCCHARLACIIA
```

Extra Credit Opportunity:

+5 Add a different option for generating the city. You still have to use the Zone class hierarchy, but you could write a different procedural generation to generate the city.

NOTE: You still must accomplish and demonstrate the specifications as written. This is an "in addition to" item.

What to turn in:

Create a zip file to submit:

- <lastname>_<firstname>_hw5.cpp
- <lastname>_CityZones.h
- <lastname>_CityZones.cpp
- Makefile

Name your zip file <lastname>_<firstname>_hw5.zip

Submit to the course canvas.

Grading of Programming Assignment

The grader will grade your program following these steps:

(1) Compile the code. If it does not compile you will receive a U on the Specifications in the Rubric

(2) The grader will read your program and give points based on the points allocated to each component, the readability of your code (organization of the code and comments), logic, inclusion of the required functions, and correctness of the implementations of each function.

Rubric:

	Levels of Achievement						
Criteria	A	B	C	D	E	U	F
Specifications Weight 50.00%	100 % The program works and meets all of the specifications.	85 % The program works and produces the correct results and displays them correctly. It also meets most of the other specifications.	75 % The program produces mostly correct results but does not display them correctly and/or missing some specifications	65 % The program produces partially correct results, display problems and/or missing specifications	35 % Program compiles and runs and attempts specifications, but several problems exist	20 % Code does not compile and run. Produces excessive incorrect results	0 % Code does not compile. Barely an attempt was made at specifications.
Code Quality Weight 20.00%	100 % Code is written clearly	85 % Code readability is less	75 % The code is readable only by someone who knows what it is supposed to be doing.	65 % Code is using single letter variables, poorly organized	35 % The code is poorly organized and very difficult to read.	20 % Code uses excessive single letter identifiers. Excessively poorly organized.	0 % Code is incomprehensible
Documentation Weight 15.00%	100 % Code is very well commented	85 % Commenting is simple but solid	75 % Commenting is severely lacking	65 % Bare minimum commenting	35 % Comments are poor	20 % Only the header comment exists identifying the student.	0 % Non existent
Efficiency Weight 15.00%	100 % The code is extremely efficient without sacrificing readability and understanding.	85 % The code is fairly efficient without sacrificing readability and understanding.	75 % The code is brute force but concise.	65 % The code is brute force and unnecessarily long.	35 % The code is huge and appears to be patched together.	20 % The code has created very poor runtimes for much simpler faster algorithms.	0 % Code is incomprehensible

Academic Integrity and Honor Code.

You are encouraged to cooperate in study group on learning the course materials. However, you may not cooperate on preparing the individual assignments. Anything that you turn in must be your own work: You must write up your own solution with your own understanding. If you use an idea that is found in a book or from other sources, or that was developed by someone else or jointly with some group, make sure you acknowledge the source and/or the names of the persons in the write-up for each problem. When you help your peers, you should never show your work to them. All assignment questions must be asked in the course discussion board. Asking assignment questions or making your assignment available in the public websites before the assignment due will be considered cheating.

*The instructor and the TA will **CAREFULLY** check any possible proliferation or plagiarism. We will use the document/program comparison tools like MOSS (Measure Of Software Similarity: http://moss.stanford.edu/) to check any assignment that you submitted for grading. The Ira A. Fulton Schools of Engineering expect all students to adhere to ASU's policy on Academic Dishonesty. These policies can be found in the Code of Student Conduct:*

`http://www.asu.edu/studentaffairs/studentlife/judicial/academic_integrity.htm`

ALL cases of cheating or plagiarism will be handed to the Dean's office. Penalties include a failing grade in the class, a note on your official transcript that shows you were punished for cheating, suspension, expulsion and revocation of already awarded degrees.
