CS 132

Project 4: Group OOP: "OOPs—I Sank the Fleet"

Group Name:

Names:

You already know how to Sink the Fleet. Now that you have the added power of *Object-Oriented Programming* to your range of programming tools, it is time to utilize it in your battle strategies as well.
In order to do this, you will need to form groups of **no less than three and no more than five**. As a group, you will need to create some new classes, as well as re-define your *Cell, ShipInfo* and *Player structs* in Object-Oriented form. You should bring in *tTextGraphics.h'* and *tTextGraphics.cpp'* just as they were before. You will need to design at least the following classes (*each class should have it's own .h and .cpp file, and all classes should be within a single namespace*):

Cship A wrapper class for **enum Ship**. Should contain **at least** the following:

properties: *(private)*
- Ship m_ship

methods: *(public)*
- a default constructor -- Initializes m_ship.
- cast operator to Ship -- returns m_ship
- print -- takes in an ostream reference. Prints out in the same format as in *project 1's printship()*
- print -- with no parameter. Prints to cout
- printName -- takes in an ostream reference. Prints out the string equivalent of the ship

Related Functions:
- **An overloaded insertion (<<) operator for an ostream and this class --** should just call print and return the ostream. *This does not need to be and should not be a friend.*
- **An overloaded extraction (>>) operator for an istream and this class --** should input either a name or a number and return the istream. *This does not need to be and should not be a friend.*

Cdirection A wrapper class for **enum Direction**. Should contain **at least** the following:

properties: *(private)*
- Direction m_direction

methods: *(public)*
- a default constructor -- Initializes m_direction.
- cast operator to Direction -- returns m_direction
- print -- takes in an ostream reference. Prints out in the same format as in *project 1 (e.g. 'H'or 'V')'*
- print -- with no parameter. Prints to cout

Related Functions:
- **An overloaded insertion (<<) operator for an ostream and this class --** should just call print and return the ostream. *This does not need to be and should not be a friend.*
- **An overloaded extraction (>>) operator for an istream and this class --** should input either a name or a number and return the istream. *This does not need to be and should not be a friend.*

Ccell OOP version of the **Cell struct** in project 1. Should contain **at least** the following:

properties: *(private)*
- unsigned short m_row
- unsigned short m_col

methods: *(public)*
- a default constructor -- Initializes m_row and m_col.
- accessors for both properties -- getRow() and getCol()

- print -- takes in an ostream reference. Prints out in the same format as in *project 1 (e.g. **'A3'br D12')***
- print -- with no parameter. Prints to cout
- inputCoordinates -- takes in an istream reference and a char ('***S***' or '***L***'). Inputs and validates the coordinates.

Related Functions:

- **An overloaded insertion (<<) operator for an ostream and this class --** should just call print and return the ostream. *This does not need to be and should not be a friend.*

CshipInfo OOP version of the **ShipInfo struct** in project 1. Should contain **at least** the following:

properties: *(private)*

- CShip m_name -- the type of ship
- CDirection m_orientation -- which direction is the ship facing?.
- CCell m_bowLocation-- coordinates of the front of the ship
- short m_piecesLeft-- the number of parts of the ship left unsunk.

methods: *(public)*

- a default constructor -- Initializes all properties.
- print -- takes in an ostream reference. Prints out in the same format as in *project 1*
- print -- with no parameter. Prints to cout.
- accessors for all properties -- getName(), getOrientation, getBowLocation() and getPiecesLeft()
- mutators for all properties -- setName(), setOrientation, setBowLocation() and setPiecesLeft()
- operator--() -- (*pre-decrement*) -- decrements m_piecesLeft and returns the current object.
- isSunk -- returns true if there are no pieces left

static constant properties: *(public)*

- static const short shipSize[6] -- the number of elements in each ship (*ignore 0*)

Related Functions:

- **An overloaded insertion (<<) operator for an ostream and this class --** should just call print and return the ostream. *This does not need to be and should not be a friend.*

CPlayer

properties: *(private)*

- unsigned short m_whichPlayer
- short m_piecesLeft -- the number of parts of the **fleet** left unsunk
- CShipInfo m_ships[6] -- an array of the ships in the fleet (ignore 0)
- char m_gridSize-- an '**L**' or '**S**' (*large or small*)
- CShip** m_gameGrid[2] -- an array of two 2-dimensional grids. Same as in *project 1*

methods: *(public)*

- a default constructor -- can take in an unsigned short *whichPlayer* and a char *gridsize*.
  validates and sets m_gridSize. sets m_whichPlayer, sets both gamegrids to NULL, calls allocateMemory(), and intializes the ships in m_ships.
- a copy constructor -- for *deep-copy*
- a destructor -- calls **deleteMemory** to clean up all memory leaks
- operator = -- for *deep-copy* assignnment
- accessors -- getWhichPlayer(), getPiecesLeft(), getGridSize()
  getCell() – takes in short whichGrid and CCell cell. Returns the Ship in the indicated cell.
- mutators
  setGridSize() – takes in a char '**L**' or '**S**' (*large or small*)
  setCell() -- takes in short whichGrid, CCell cell, and CShip ship. Set the proper cell to ship.
- printGrid -- takes in an ostream reference and short whichGrid . Prints out the appropriate grid.
- getGrid -- takes in string filename. inputs a grid from a file.
- saveGrid -- inputs and validates filename and prints the gridSize, m_ships and grid[0] to the file. Same as in *project 1*
- setShips -- lets the player set up the grid
- hitShip -- takes in a CShip ship. Decrements the pieces left for the ship and the fleet.
- isValidLocation -- takes in a short whichShip (*the index number of which ship in the array*).
  Returns true if the bow location is valid --**i.e.**, will not put the ship out-of-bounds or across another ship.

- operator[] const -- takes in a short index. Validates index and returns m_ships[index]
- operator--() -- decrements m_piecesLeft and returns the current object.

methods: *(private)*

- allocateMemory -- allocates and initializes the game grids.
- deleteMemory -- clean-up

CSinkTheFleet

properties: *(private)*

- CPlayer m_players[2] -- the two players
- char m_gridSize – an **'L'** or **'S'** *(large or small)*

methods: *(public)*

- a default constructor -- Has char parameter for gridSize. Initializes m_gridSize and the two players.
- play -- plays the game, the same as in *project 1*. Returns the index of the winning player.
- operator[] -- takes in a short index. Validates index and returns m_players[index]
- operator[] const -- takes in a short index. Validates index and returns m_players[index]
- static header -- same as in *project 1*
- static endbox – same as in *project 1*

The program should instantiate a CSinkThe Fleet object. It should then, using the provided classes and methods, get the game size, set up the two players as in *project 1,* and call the *play* method. The game should then announce the winner, and ask if the players wish another game.

The program should be fully planned *in advance, using object-oriented design*. It should be well documented, and work correctly.

Make sure to test your program with ships extending out-of-bounds, overlapping ships, reading in from files, firing out of bounds and firing twice at the same spot. Don't forget to test for invalid input.

Points Possible: 100

Extra Credit: 10 points If you get the program together and still have the energy, you may gain up to 10 points extra credit.
In the CPlayer class, add in the method autoSetShips, which will randomly set the locations of the five ships and place them on the grid. The program should then give this as a third option for setting up the ships.

Deliverables:

Physical:

- For this project only: none.

Electronic:

- All .h, .cpp, .exe(Release Version) zipped together. Do not use rar or any archive format other than zip. Rename the file: "<YourGroupName>_p4.zip". Submit one file per group.
- Sample Output (as .rtf -- run the program, copy the window using <*Alt*|*PrtScn*>, paste into Paint, invert colors *(*<*Ctrl*|*Shift*|*I*>), copy, open Wordpad, save.)
- A simple test plan including explanations of any discrepancies and reasons for each test. Show actual input and ALL values output as well as ALL expected output. Test each possible action. Save as .xls, xlsx, .doc or .docx file
- Submit this single zip file by going to canvas, select this class, select the Assignment tab on the left, select Assignment 4, select the submission tab at the top right, find the file, and Submit.

  Due: **Section A:** *Monday, March 14, 2016 9:30 a.m. (beginning of lab)*
  Section B: *Thursday, March 10, 2016 4:30 p.m. (beginning of lab)*
  Informal presentations will be given at this time