BLG252E - Object Oriented Programming

The 2nd Assignment

14.04.2014 – 29.04.2014, 23.00

In this assignment, you'll design and implement necessary classes for the simulation of the classical ***"Battleships"*** game. ***"Battleships"*** is a two player game which is played on a 10x10 grid board. Considering the object oriented programming approach, please go through the descriptions and concepts below.

Ships

There are 4 different types of ships in the game. These ships are submarine (S), destroyer (D), cruiser (C) and battleship (B). They can be placed on the grid-based game board horizontally or vertically. The only difference between these ships is their sizes. They cover 1, 2, 3 and 4 squares on the grid board respectively. Any square of a ship can take a hit. If all squares of a ship are hit, then the ship is sank.

Players

Each player in the game has 10 ships containing 4 submarines, 3 destroyers, 2 cruisers and a battleship. Players place their ships on their own 10x10 grid game board, and attack to each other in turn. To attack the opponent, player says a game board coordinate such as 'H7' which states the 7th column of the 8th (H) row. If there is a ship piece in that coordinate, the ship is hit. If all pieces of the ship is hit, then it is sank. The opponent should warn the other player about the success or fail of the attack. Player whose all ships are sank loses the game.

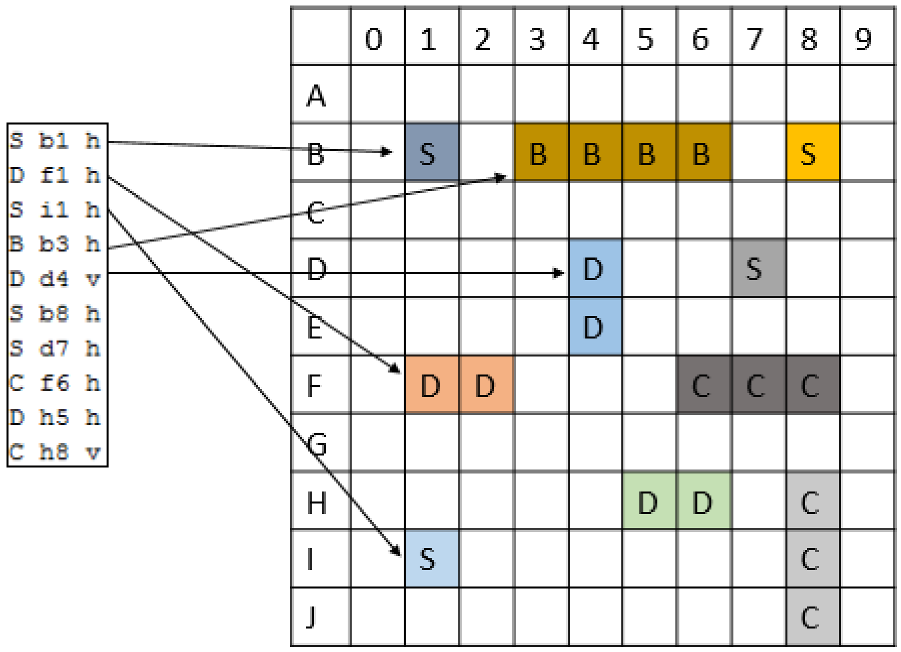
Game Implementation

To model the players, ***"Side"*** class is designed as follows. You should implement its methods with respect to the descriptions below and usages in the main function.

```
class Side {
    private:
        Ship **list;
        string attack_history[100];
        int counter;
    public:
        Side(char *);
        void print() const;
        bool defeated() const;
        char damage(char, int);
        string attack();
        ~Side();
};
```

"list" is the array of pointers to the ***"Ship"*** classes. Although, the elements of ***"list"*** are pointers to ***"Ship"*** class, they should point different types of ships. ***"attack_history"*** is for storing previous attacks of the player, and ***"counter"*** is the index of the last element of this array.

Constructor of this class takes a file name as parameter (char*). Ships are placed on the board according to this input file. An example organization of input file and game board is given as follows.



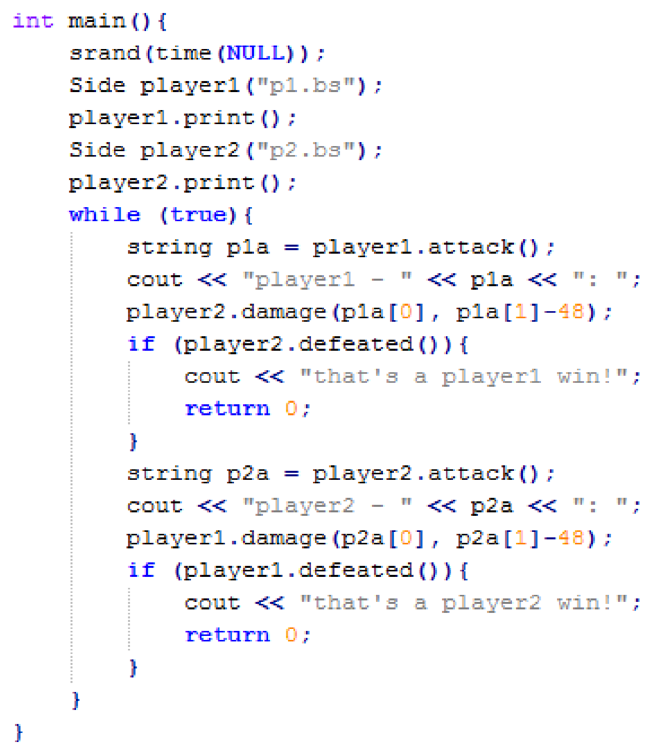
```
S b1 h
D f1 h
S i1 h
B b3 h
D d4 v
S b8 h
S d7 h
C f6 h
D h5 h
C h8 v
```

	0	1	2	3	4	5	6	7	8	9
A										
B		S		B	B	B	B		S	
C										
D					D			S		
E					D					
F		D	D				C	C	C	
G										
H						D	D		C	
I		S							C	
J									C	

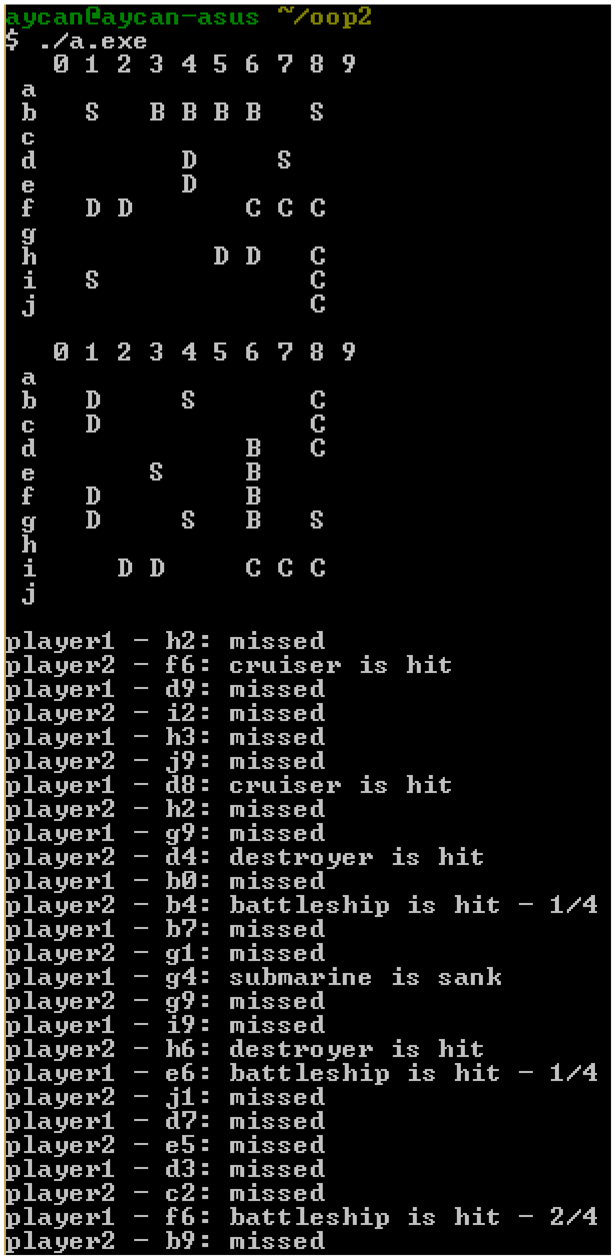
As stated above, each "Ship*" element in the array of "list" should be defined one of the following four types; submarine, destroyer, cruiser and battleship. In fact, there isn't any "Ship" class in use (see abstract classes in your lecture notes).

"print" function prints the game board to the screen as in the figure above. ***"defeated"*** function checks the player, whether he loses the game or not. ***"damage"*** function takes a char and an integer as its parameters which represent the board coordinate attacked by the opponent. ***"damage"*** function checks the given coordinate if there's a ship at that location and performs necessary operations, more specifically it controls the coordinates and if there's a ship at that location prints the related message to the screen. Additionally, only for battleships, "damage" function prints number of damaged parts (see sample output). Finally, ***"attack"*** function generates a board coordinate randomly, e.g., H6 (a two-length string with a letter and a number). Any coordinate that is attacked before should not be generated again. That's why, you should use the variable ***"attack_history"*** to store previous attacks. Usages of these functions can be seen clearly from the main function below.

According to the given code pieces, you should implement the methods of ***"Side"*** class and design ***"Ship"*** classes to finish the whole assignment. Sample output of the program is as follows (output continues, this is just the beginning of it).



```cpp
int main(){
    srand(time(NULL));
    Side player1("p1.bs");
    player1.print();
    Side player2("p2.bs");
    player2.print();
    while (true){
        string p1a = player1.attack();
        cout << "player1 - " << p1a << ": ";
        player2.damage(p1a[0], p1a[1]-48);
        if (player2.defeated()){
            cout << "that's a player1 win!";
            return 0;
        }
        string p2a = player2.attack();
        cout << "player2 - " << p2a << ": ";
        player1.damage(p2a[0], p2a[1]-48);
        if (player1.defeated()){
            cout << "that's a player2 win!";
            return 0;
        }
    }
}
```



```
aycan@aycan-asus ~/oop2
$ ./a.exe
   0 1 2 3 4 5 6 7 8 9
 a
 b   S   B B B B   S
 c
 d         D     S
 e         D
 f   D D       C C C
 g
 h           D D   C
 i   S             C
 j                 C

   0 1 2 3 4 5 6 7 8 9
 a
 b   D     S       C
 c   D             C
 d             B   C
 e       S     B
 f   D         B
 g   D     S   B   S
 h
 i     D D     C C C
 j

player1 - h2: missed
player2 - f6: cruiser is hit
player1 - d9: missed
player2 - i2: missed
player1 - h3: missed
player2 - j9: missed
player1 - d8: cruiser is hit
player2 - h2: missed
player1 - g9: missed
player2 - d4: destroyer is hit
player1 - b0: missed
player2 - b4: battleship is hit - 1/4
player1 - b7: missed
player2 - g1: missed
player1 - g4: submarine is sank
player2 - g9: missed
player1 - i9: missed
player2 - h6: destroyer is hit
player1 - e6: battleship is hit - 1/4
player2 - j1: missed
player1 - d7: missed
player2 - e5: missed
player1 - d3: missed
player2 - c2: missed
player1 - f6: battleship is hit - 2/4
player2 - b9: missed
```

Hints

- “Ship” class may contain common variables of different types of ships such as coordinate(s), position, etc. (it is all up to you)
- Obviously, “damage” and “print” functions in “Side” class use other functions that belong to subclasses of “Ship” class
- In ship class and its sub classes, you may or may not define variables dynamically with memory allocation.
- Use ASCII arithmetic to calculate coordinates (each character is also an integer in [0,255]).

Implementation Notes

- Implement only necessary methods.
- Do not write extra constructors or destructors.
- Be careful with the methods/attributes which are supposed to be *constant, static*.
- Do not use *friend* relations in your homework and make sure that all of your class attributes are *private or protected* with respect to the design.
- Consider OOP approach in your design. Implementations that do not contain inheritance and polymorphism when necessary will not be graded.
- For any questions about the assignment, contact Ahmet Aycan Atak directly (office no:4316) or via mail (ataka@itu.edu.tr)

Submission Notes

- Submissions are through Ninova system and has a **strict deadline**, no assignments submitted after deadline is accepted.
- Source codes should be uploaded as one compressed file named after your student number as "*student_number*.zip"
- You should implement your program in C++ programming language.
- Be sure that your program works and produces reasonable and expected results.
- Be sure to include clear code comments.
- This is *not* a group assignment and getting involved in any kind of cheating induces negative grades.
- For any problems with the assignment, contact Ahmet Aycan Atak directly (office no:4316) or via mail (ataka@itu.edu.tr)