```cpp
//battleship function definitions
//Justin Mckenna
//12/17/2017

#include "battleship.h"
#include <iostream>
#include <string>
#include <fstream>

using std::cout; using std::cin; using std::string;
using std::endl; using std::ifstream; using std::ofstream;

//sets default location to (*-1)
Location::Location() {
	x = -1;
	y = '*';
}

//picks a random location on field
void Location::pick() {
	x = rand() % fieldSize + 1;
	switch (rand() % fieldSize + 1) {
	case 1:
		y = 'a';
		break;
	case 2:
		y = 'b';
		break;
	case 3:
		y = 'c';
		break;
	case 4:
		y = 'd';
		break;
	case 5:
		y = 'e';
		break;
	default:
		break;
	}
}

//user fires a shot
void Location::fire() {
	string shot;
	cin >> shot;
	x = shot[1] - '0';
	y = shot[0];
}

//prints location
void Location::print() const {
	cout << y << x << " ";
}

//checks if locations are the same
bool compare(const Location& a, const Location& b) {
	return a.x == b.x && a.y == b.y;
```

```cpp
}

//sets sunk to false
Ship::Ship() {
    sunk = false;
}

//ship is deployed at specified location
void Ship::setLocation(const Location& spot) {
    loc = spot;
}

//checks if ship location matches specified location
bool Ship::match(const Location& spot) const {
    return compare(loc, spot);
}

//changes sunk to true
void Ship::sink() {
    sunk = true;
}

//prints location and sunk status of ship
void Ship::printShip() const {
    loc.print();
    if (isSunk())
        cout << "Sunk ";
    else
        cout << "Up ";
}

//returns index of ship that matches location, -1 if none match
int Fleet::check(const Location & spot) const {
    for (int i = 0; i < fleetSize; ++i) {
        if (ships[i].match(spot))
            return i;
    }
    return -1;
}

//shows ships on field grid
void Fleet::showShips(int field[][fieldSize]) {
    int numDepShips = 0;
    while (numDepShips < fleetSize) {
        int fieldX = ships[numDepShips].loc.x;
        int fieldY;
        switch (ships[numDepShips].loc.y) {
        case 'a':
            fieldY = 0;
            break;
        case 'b':
            fieldY = 1;
            break;
        case 'c':
            fieldY = 2;
            break;
        case 'd':
            fieldY = 3;
```

```
			break;
		case 'e':
			fieldY = 4;
			break;
		default:
			break;
		}
		field[fieldX - 1][fieldY] = 3;
		numDepShips += 1;
	}
}

//deploys ships to random locations within ocean
void Fleet::deployFleet() {
	int numDepShips = 0;
	while (numDepShips < fleetSize) {
		Location random;
		random.pick();
		if (check(random) == -1) {
			ships[numDepShips].setLocation(random);
			numDepShips += 1;
		}
	}
}

//checks if at least one ship is not suck
bool Fleet::operational() const {
	for (int i = 0; i < fleetSize; ++i) {
		if (!ships[i].isSunk())
			return true;
	}
	return false;
}

//sinks ship if specific location matches a ship location
bool Fleet::isHitNSink(const Location & spot, int field[][fieldSize]) {
	int fieldX = spot.x;
	int fieldY;
	switch (spot.y) {
	case 'a':
		fieldY = 0;
		break;
	case 'b':
		fieldY = 1;
		break;
	case 'c':
		fieldY = 2;
		break;
	case 'd':
		fieldY = 3;
		break;
	case 'e':
		fieldY = 4;
		break;
	default:
		break;
	}
	for (int i = 0; i < fleetSize; ++i) {
        if (ships[i].match(spot)) {
            ships[i].sink();
            field[fieldX - 1][fieldY] = 2;
            return true;
        }
    }
    field[fieldX - 1][fieldY] = 1;
    return false;
}

//prints fleet
void Fleet::printFleet() const {
    for (int i = 0; i < fleetSize; ++i) {
        ships[i].printShip();
    }
}


//prints field grid
void printField(int field[][fieldSize]) {
    for (int i = 0; i < fieldSize + 1; ++i) {
        for (int j = 0; j < fieldSize + 1; ++j) {
            if (i == 0 && j == 0)
                cout << " ";
            else if (i == 0)
                switch (j) {
                case 1:
                    cout << "a";
                    break;
                case 2:
                    cout << "b";
                    break;
                case 3:
                    cout << "c";
                    break;
                case 4:
                    cout << "d";
                    break;
                case 5:
                    cout << "e";
                    break;
                default:
                    break;
                }
            else if (j == 0)
                cout << i;
            else {
                if (field[i - 1][j - 1] == 0)
                    cout << " ";
                else if (field[i - 1][j - 1] == 1)
                    cout << "O";
                else if (field[i - 1][j - 1] == 2)
                    cout << "X";
                else
                    cout << "S";
            }

        }
```

```cpp
		cout << endl;
	}
}

//assigns every element of shots to false
void initialize(int field[][fieldSize]) {
	for (int i = 0; i < fieldSize; ++i)
		for (int j = 0; j < fieldSize; ++j)
			field[i][j] = 0;
}

//tracks highscores
void getScore(int numTurns) {
	ofstream fout("highScores.txt");
	if (!fout.fail()) {
		cout << "Input your name: ";
		string name;
		cin >> name;
		fout << numTurns << " " << name;
		fout.close();
	}
}

//check highscore
bool bestScore(int &numTurns) {
	int i;
	ifstream fin("highScores.txt");
	if (!fin.fail()) {
		fin >> i;
		if (i > numTurns) {
			cout << "You have the best score!" << endl;
			fin.close();
			return true;
		} else {
			cout << "You don't have the best score!" << endl;
			fin.close();
			return false;
		}
	}
}

//prints highscores
void printScore() {
	cout << "Best Score: ";
	ifstream fin("highScores.txt");
	if (!fin.fail()) {
		string line;
		while (getline(fin, line))
			cout << line;
	}
	cout << endl;
}
```