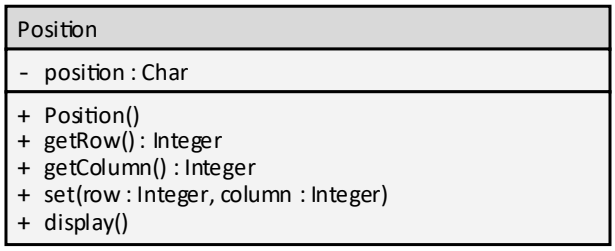
CSE 230 Problem Set 08

Problem 24.1: Fragile

Consider a Position class representing a position on a chess board. Here the internal representation of the position is a single character. Note that a character has 256 possible values, but a chess board has 64 possible values. There are therefore plenty of bits to represent this position. To accomplish the mapping between the row and column values and the internal representation, the following code is provided:



Pseudocode

```
getRow()
   RETURN position / 8

getCol()
   RETURN position % 8

set(row, column)
   position ← row * 8 + column
```

Implement this class in C++. When you are finished, establish that your implementation has fragile robustness:

```
class Position {
private:
   char position;

public:
   Position();
   int getRow() {return int(position) / 8;}
   int getColumn() { return int(position) % 8;}
   void set(int row, int column);
   void display();
};

void set(int row, int column) {
   if (8 > row > -1 && 8 > column > -1)  {
      position = char((row * 8) + column);
```

```
    }

}

void display() {
    std::cout << "Row: " << getRow() << "\n";
    std::cout << "Column: " << getColumn() << "\n";
}
```

Rationale This class has not been tested thoroughly, and only really been looked over to check for proper functionality. It has fragile robustness right now.

Problem 24.2: Tested

From the Position class of Problem 24.1, create the assurances necessary to establish that the class has tested robustness. If any bugs are found in this process, please fix them and provide the new class definition.

Code of driver

```
void testPosition(int inputRows[], int inputColumns[], int outputRows[], int outputColumns[],int
testNum) {
   Position position = Position();
   std::cout << "Beginning testing\n";
   for (int index = 0; index < testNum; index ++) {
      int row = inputRows[index];
      int column = inputColumns[index];
      position.set(row,column);
      assert(position.getRow() == outputRows[index]);
      assert(("Error with column\n",position.getColumn() == outputColumns[index]));
   }
   std::cout << "Testing Ended Successfully\n";
}
```

Rationale This class is now tested. It properly sets and returns information that is required as part of the position class. It hasn't gone into super thorough testing, but it has begun to be tested.

Code of Position Class

```
class Position {
private:
   char position;

public:
   Position() {return;};
   int getRow() {return int(position) / 8;}
   int getColumn() { return int(position) % 8;}
   void set(int row, int column);
   void display();
```

```
};

void Position::set(int row, int column) {
    if ((8 > row && row > -1) && (8 > column && column > -1))  {
        position = char((row * 8) + column);
    }

}

void Position::display() {
    std::cout << "Row: " << getRow() << "\n";
    std::cout << "Column: " << getColumn() << "\n";
}
```

Problem 24.3: Strong

From the Position class of Problem 24.1 and 24.2, create the assurances necessary to establish that the class has strong robustness. If any bugs are found in this process, please fix them and provide the new class definition.

Hint: You may need to create a simple test document to solve this problem and write some simple automation.

Test cases

Name	Input	Output
Normal	0,1	0,1
Normal	7,3	7,1
Letter in row	'a',2	0,2
Negative row	-1,7	0,7
Too high row	9,6	0,6
Too high row	12,2	0,2
Normal	1,0	1,0
Normal	2,7	2,7
Letter in column	3,'C',	3,0
Negative column	6,-5	6,0
Too high column	5,15	5,0
Too high column	2,10	2,0

Code of driver

```
void testPosition(auto inputRows[], auto inputColumns[], auto outputRows[],
auto outputColumns[],int testNum) {
    Position position = Position();
    std::cout << "Beginning testing\n";
    for (int index = 0; index < testNum; index ++) {
        int row = inputRows[index];
        int column = inputColumns[index];
        position.set(row,column);
        if (position.getRow() == outputRows[index]) {
            std::cout << "Row Test " << index+1 << " Passed\n";
        } else {
            std::cout << "Row Test " << index+1 << " Failed\n Expected
Output: " << outputRows[index] << ", actual: " << position.getRow() << "\n";
        }

        if (position.getColumn() == outputColumns[index]) {

            std::cout << "Column Test " << index+1 << " Passed\n";
        } else {
```

```
            std::cout << "Column Test " << index+1 << " Failed\n Expected
Output: " << outputColumns[index] << ", actual: " << position.getColumn() <<
"\n";
        }
    }
    std::cout << "Testing Ended\n";
}
```

Problem 24.4: Resilient

From the Position class of Problem 24.1 – Problem 24.3, create the assurances necessary to establish that the class has resilient robustness. Provide only 3 test functions. If any bugs are found in this process, please fix them and present the updated class definition.

```
 Code of tests
void testPosition(auto inputRows[], auto inputColumns[], auto outputRows[],
auto outputColumns[],int testNum) {
    Position position = Position();
    std::cout << "Beginning testing\n";
    for (int index = 0; index < testNum; index ++) {
        int row = inputRows[index];
        int column = inputColumns[index];
        position.set(row,column);
        if (position.getRow() == outputRows[index]) {
            std::cout << "Row Test " << index+1 << " Passed\n";
        } else {
            std::cout << "Row Test " << index+1 << " Failed\n Expected
Output: " << outputRows[index] << ", actual: " << position.getRow() << "\n";
        }

        if (position.getColumn() == outputColumns[index]) {

            std::cout << "Column Test " << index+1 << " Passed\n";
        } else {
            std::cout << "Column Test " << index+1 << " Failed\n Expected
Output: " << outputColumns[index] << ", actual: " << position.getColumn() <<
"\n";
        }
    }
    std::cout << "Testing Ended\n";
}

void Position::testASCII(int inputRows[], int inputColumns[], char
outputASCII[], int testNum) {

    std::cout << "Beginning testing\n";
    for (int index = 0; index < testNum; index++) {
        int row = inputRows[index];
        int column = inputColumns[index];
        set(row, column);

        if (position == outputASCII[index]) {
            std::cout << "ASCII Test " << index + 1 << " Passed\n";
        }
        else {
            std::cout << "ASCII Test " << index + 1 << " Failed\n Expected
Output: " << outputASCII[index] << ", actual: " << position << "\n";
        }
```

```
    }
    std::cout << "Testing Ended\n";
}

void Position::testIntPosition(int inputRows[], int inputColumns[], int
outputs[], int testNum) {
    for (int index= 0; index < testNum; index++ ) {
        int row = inputRows[index];
        int column = inputColumns[index];
        set(row,column);

        if (int(position) == outputs[index]) {
            std::cout << "Position Integer Test " << index + 1 << "
Passed\n";
        }
        else {
            std::cout << "Position Integer Test " << index + 1 << " Failed\n
Expected Output: " << outputs[index] << ", actual: " << int(position) <<
"\n";
        }
        }
    }
```

Class Code:

```
//
//  Position.hpp
//  robustness practice
//
//  Created by Jacob on 6/8/23.
//

#ifndef Position_hpp
#define Position_hpp

#include <stdio.h>
#include <iostream>

class Position {
private:
    char position;

public:
    Position() {return;};
```

```
    int getRow() {return (int(position) - 33) / 8;}
    int getColumn() { return (int(position) - 33) % 8;}
    void set(int row, int column);
    void display();
    void testASCII(int inputRows[], int inputColumns[], char outputASCII[],
int testNum);
    void testIntPosition(int inputRows[], int inputColumns[], int outputs[],
int testNum);
};

void Position::set(int row, int column) {
    if ((8 > row && row > -1) && (8 > column && column > -1))  {
        position = char(((row * 8) + column + 33)) ;
    } else {
        position = char(33);
    }

}

void Position::display() {

    std::cout << "Row: " << getRow() << "\n";
    std::cout << "Column: " << getColumn() << "\n";
}

void Position::testASCII(int inputRows[], int inputColumns[], char
outputASCII[], int testNum) {

    std::cout << "Beginning testing\n";
    for (int index = 0; index < testNum; index++) {
        int row = inputRows[index];
        int column = inputColumns[index];
        set(row, column);

        if (position == outputASCII[index]) {
            std::cout << "ASCII Test " << index + 1 << " Passed\n";
        }
        else {
            std::cout << "ASCII Test " << index + 1 << " Failed\n Expected
Output: " << outputASCII[index] << ", actual: " << position << "\n";
        }

    }
    std::cout << "Testing Ended\n";
}

void Position::testIntPosition(int inputRows[], int inputColumns[], int
outputs[], int testNum) {
    for (int index= 0; index < testNum; index++ ) {
        int row = inputRows[index];
```

```
        int column = inputColumns[index];
        set(row,column);

        if (int(position) == outputs[index]) {
            std::cout << "Position Integer Test " << index + 1 << "
Passed\n";
        }
        else {
            std::cout << "Position Integer Test " << index + 1 << " Failed\n
Expected Output: " << outputs[index] << ", actual: " << int(position) <<
"\n";
        }
        }
    }


#endif /* Position_hpp */
```