CSE 230 Problem Set 05

# Problem 22.1: 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:



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

#pragma once

#include <iostream>

using namespace std;

class Position

{

private:

char position;

public:

Position()

{

position = 0;

}

int getRow() { return position / 8; }

int getCol() { return position % 8; }

void set(int row, int column)

{

position = row \* 8 + column;

}

void display()

{

cout << "Position: " << getRow() + 1;

switch (getCol())

{

case 0: cout << "A\n"; break;

case 1: cout << "B\n"; break;

case 2: cout << "C\n"; break;

case 3: cout << "D\n"; break;

case 4: cout << "E\n"; break;

case 5: cout << "F\n"; break;

case 6: cout << "G\n"; break;

case 7: cout << "H\n"; break;

default:

break;

}

}

};

The thing that is hard to guarantee is not having correct input for the setter. Also, the design can be updated to implement `unsigned char` instead of char. There won’t be a memory improvement, however this will prevent negative values…which don`t really make sense in this situation. We can at least include this, as well as extra error handling in the setter.

# Problem 22.2: Tested

From the Position class of Problem 22.1, 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.

int main() {

// test a valid position first

cout << "Test King position\n";

Position king; // create position

king.set(0, 4); // set position functionality

king.display(); // test display functionality

// test getters functionality

cout << "King row/column indexes: "

<< king.getRow() << " " << king.getCol() << endl;

// test invalid positions

cout << "\nInvalid position no. 1\n";

Position invalid;

invalid.set(-1, -1);

invalid.display();

cout << "\nInvalid position no. 2\n";

Position invalid2;

invalid2.set(64, 64);

invalid2.display();

}

Tested robustness is ensured, just with some bugs. Invalid positions don`t throw errors in the setter. Also, let`s implement the unsigned char instead of char for the position.

#pragma once

#include <iostream>

using namespace std;

class Position

{

private:

unsigned char position;

public:

Position()

{

position = 0;

}

int getRow() { return position / 8; }

int getCol() { return position % 8; }

void set(int row, int column)

{

if (row < 0 || column < 0 || row > 0 || column > 0) {

cout << "Range error: cannot set this chess \

piece in the specified position.\n";

return;

}

position = row \* 8 + column;

}

void display()

{

cout << "Position: " << getRow() + 1;

switch (getCol())

{

case 0: cout << "A\n"; break;

case 1: cout << "B\n"; break;

case 2: cout << "C\n"; break;

case 3: cout << "D\n"; break;

case 4: cout << "E\n"; break;

case 5: cout << "F\n"; break;

case 6: cout << "G\n"; break;

case 7: cout << "H\n"; break;

default:

break;

}

}

};

# Problem 22.3: Strong

From the Position class of Problem 22.1 and 22.2, 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

Code of driver

# Problem 22.4: Resilient

From the Position class of Problem 22.1 – Problem 22.3, 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