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:

Position
- position : Char

+ Position()
+ getRow() : Integer
+ getColumn() : Integer
+ set(row : Integer, column : Integer)
+ display()

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:

```
Code
#include <iostream>
using namespace std;

class Position
{
private:
   char pos; // Stores the position as a single character.

public:
   Position(int row, int col) { pos = (row * 8) + col; }

   // Decode the position back into row and column.
   int getRow() const { return pos / 8; }
   int getCol() const { return pos % 8; }

   void set(int row, int column) { pos = (row * 8) + column; }

   // Print the position
   void display() const
   {
      cout << "Row: " << getRow() << ", Col: " << getCol() << endl;
   }
};
```

```
// Driver Code
int main()
{
   Position p(3, 5);
   p.display(); // Expected output: Row: 3, Col: 5
   return 0;
}
```

Rationale: The code functions under expected conditions but has no boundary checks or error handling conditions so it will crash at any invalid input.

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
// Driver Code with Tests
int main()
{
   try
   {
      Position p(3, 5);
      p.print(); // Valid position

      Position invalidPos(-1, 2); // Should throw an exception
   }

   catch (const std::exception &e)
   {
      std::cerr << "Exception: " << e.what() << std::endl;
   }

   return 0;
}
```

Rationale: Tests just enough to make sure it does not crash but does not test any edge cases or weird scenarios.

```
Code of Position Class
#include <iostream>
#include <stdexcept>

class Position
{
   private:
   char pos; // Stores the position as a single character.

   public:
   // Constructor with validation
   Position(int row, int col)
   {
      if (row < 0 || row >= 8 || col < 0 || col >= 8)
      {
         throw std::out_of_range("Invalid chessboard position");
      }
      pos = static_cast<char>((row * 8) + col);
   }

   int getRow() const { return pos / 8; }
```

```
    int getCol() const { return pos % 8; }

    void print() const
    {
        std::cout << "Row: " << getRow() << ", Col: " << getCol() << std::endl;
    }
};
```

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 uses the assert library to systematically test valid, invalid and edge cases to ensure the program handles each of them without failure.

```
Code of driver
#include <iostream>
#include <stdexcept>
#include <cassert>

// Automated test cases
void testValidPositions()
{
   Position p(4, 7);
   assert(p.getRow() == 4);
   assert(p.getCol() == 7);
}

void testInvalidPositions()
{
   try
   {
      Position p(-1, 3);
      assert(false); // Should not reach this line
   }
   catch (const std::out_of_range &)
   {
      assert(true);
   }
}

void testEdgeCases()
{
   Position p1(0, 0);
   assert(p1.getRow() == 0);
   assert(p1.getCol() == 0);

   Position p2(7, 7);
   assert(p2.getRow() == 7);
   assert(p2.getCol() == 7);
}

int main()
{
    testValidPositions();
    testInvalidPositions();
    testEdgeCases();

    std::cout << "All tests passed!" << std::endl;
    return 0;
}
```

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
#include <iostream>
#include <stdexcept>
#include <cassert>

using namespace std;

class Position
{
private:
   char pos; // Stores the position as a single character.

public:
   // Constructor with validation
   Position(int row, int column)
   {
      if (row < 0 || row >= 8 || column < 0 || column >= 8)
      {
         throw out_of_range("Invalid chessboard position");
      }
      pos = static_cast<char>((row * 8) + column);
   }

   int getRow() const { return pos / 8; }
   int getCol() const { return pos % 8; }

   void set(int row, int column)
   {
      if (row < 0 || row >= 8 || column < 0 || column >= 8)
      {
         throw out_of_range("Invalid chessboard position");
      }
      pos = static_cast<char>((row * 8) + column);
   }

   void print() const
   {
      cout << "Row: " << getRow() << ", Col: " << getCol() << endl;
   }
};

// Test Functions
void testValidPosition()
{
   try
   {
      Position p(2, 3);
      assert(p.getRow() == 2);
      assert(p.getCol() == 3);
      cout << "testValidPosition passed.\n";
   }
   catch (...)
   {
      cout << "testValidPosition failed! Exception thrown unexpectedly.\n";
   }
}

void testOutOfBoundsPosition()
{
   try
   {
      Position p(8, 8); // Invalid case
      assert(false); // Should never reach here
   }
   catch (const out_of_range &)
   {
      cout << "testOutOfBoundsPosition passed.\n";
   }
   catch (...)
   {
      cout << "testOutOfBoundsPosition failed! Unexpected exception type.\n";
   }
}

void testLowerBoundPosition()
{
   try
   {
      Position p(0, 0);
      assert(p.getRow() == 0);
      assert(p.getCol() == 0);
      cout << "testLowerBoundPosition passed.\n";
   }
   catch (...)
   {
      cout << "testLowerBoundPosition failed! Exception thrown
unexpectedly.\n";
   }
}
int main()
{
   testValidPosition();
   testOutOfBoundsPosition();
   testLowerBoundPosition();
```

```
    cout << "All resilience tests passed!" << endl;
    return 0;
}
```