CSE 230 Problem Set 10

Problem 26.2: Step 1

Complete step 1 (and the 4 sub-steps) of the TDD process for a method in a class which stores a position on a chess board:

> A chess board consists of 64 locations: 8 rows and 8 columns. Every column has a letter (a-h) and every row has a number (1-8). The user can use upper-case or lower-case letters and can even get the order mixed up. Thus, “c2” means the same thing as “2C” which is position 10. This is for the `Coordinate::set(const char *input)` method.

	a	b	c	d	e	f	g	h
8	56	57	58	59	60	61	62	63
7	48	49	50	51	52	53	54	55
6	40	41	42	43	44	45	46	47
5	32	33	34	35	36	37	38	39
4	24	25	26	27	28	29	30	31
3	16	17	18	19	20	21	22	23
2	8	9	10	11	12	13	14	15
1	0	1	2	3	4	5	6	7

Complete step 1: the requirements.

Requirements
Valid Chess Board Coordinates • The method should accept a valid chess board position in the format of a letter (A-H or a-h) and a number (1-8) in any order. • Examples of valid inputs: "A1", "h8", "3B", "g7", "2d".
Case Insensitivity • The method should accept both uppercase and lowercase letters. • Example: "C2" and "c2" should be treated the same.
Mixed Order Handling • The method should correctly interpret inputs even if the row and column order is reversed. • Example: "2C" should be treated the same as "C2".
Conversion to Board position Index • The method should store the correct position based on the input: • Column letters ('a'–'h' or 'A'–'H') correspond to indices (0–7). • Row numbers ('1'–'8') correspond to indices (0–7) from bottom to top. • The final board position index is calculated as: • Index = (row – 1) × 8 + column • Example: "C2" (or "2C") should be converted to index 10.

Error Handling Requirements • Inputs that do not contain exactly one letter (A-H or a-h) and one number (1-8) should be rejected. • Example invalid inputs: • "I3" (Invalid letter) • "9A" (Invalid number) • "22" (No letter) • "CC3" (More than one letter) • "B" (Only one character)
Handle Null or Empty Input • If nullptr or an empty string "" is provided, the method should reject it or return an error.
Ignore Extra Whitespace • If spaces exist in the input (e.g., " C2 " or " 3 D"), they should be ignored.

Problem 26.2: Step 2-5 for Bottom Left Corner

Complete step 2-5 of the TDD process for the “a1” test case:

Step 2: Write the test.

```
#include <iostream>
#include "Coordinate.h"  // Include the class definition

void testSetBottomLeftCorner()
{
   Coordinate coord;
   coord.set("a1");

   if (coord.getIndex() == 0)
   {
        std::cout << "PASS: 'a1' correctly maps to index 0\n";
   }

   else
   {
        std::cout << "FAIL: 'a1' mapped to " << coord.getIndex() << " instead of 0\n";
   }
}

int main()
{
   testSetBottomLeftCorner();
   return 0;
}
```

Step 3: Run the test (show the output here):

```
 FAIL: 'a1' mapped to -1 instead of 0
```

Step 4: Write the code:

```
#include "Coordinate.h"
#include <cctype>  // for std::tolower
#include <cstring> // for strlen

void Coordinate::set(const char *input)
{
   if (input == nullptr || std::strlen(input) < 2 || std::strlen(input) > 3)
   {
      valid = false;
      return;
   }

   char letter = '\0';
   char number = '\0';

   // Identify the letter (A-H) and number (1-8) in the input
   for (size_t i = 0; input[i] != '\0'; ++i)
   {
      if (std::isalpha(input[i])) letter = std::tolower(input[i]);
      if (std::isdigit(input[i])) number = input[i];
   }

   // Validate the parsed values
   if (letter < 'a' || letter > 'h' || number < '1' || number > '8')
   {
      valid = false;
      return;
   }
```

```
   // Convert to board index (0-63)
   int col = letter - 'a';   // 'a' -> 0, 'h' -> 7
   int row = number - '1';   // '1' -> 0, '8' -> 7

   index = row * 8 + col;    // Compute 0-based index
   valid = true;
}

#ifndef COORDINATE_H
#define COORDINATE_H

class Coordinate
{
private:
   int index;
   bool valid;

public:
   Coordinate() : index(-1), valid(false) {}

   void set(const char *input);
   int getIndex() const { return index; }
   bool isValid() const { return valid; }
};

#endif // COORDINATE_H
```

Step 5: Refactor:

```
 PASS: 'a1' correctly maps to index 0
```

Problem 26.3: Step 2-5 for Bottom Middle

Complete step 2-5 of the TDD process for the “c1” test case:

Step 2: Write the test.

```
#include <iostream>
#include "Coordinate.h"  // Include the Coordinate class

void testSetBottomMiddle()
{
   Coordinate coord;
   coord.set("c1");

   if (coord.getIndex() == 2)
   {
      std::cout << "PASS: 'c1' correctly maps to index 2\n";
   }

   else
   {
      std::cout << "FAIL: 'c1' mapped to " << coord.getIndex() << " instead of 2\n";
   }
}

int main()
{
   testSetBottomMiddle();
   return 0;
}
```

Step 3: Run the test (show the output here):

```
 FAIL: 'c1' mapped to -1 instead of 2
```

Step 4: Write the code:

```
#ifndef COORDINATE_H
#define COORDINATE_H

class Coordinate
{
private:
   int index;
   bool valid;

public:
   Coordinate() : index(-1), valid(false) {}

   void set(const char *input);
   int getIndex() const { return index; }
   bool isValid() const { return valid; }
};

#endif // COORDINATE_H
```

Step 5: Refactor:

```
 PASS: 'c1' correctly maps to index 2
```

Problem 26.4: Step 2-5 The rest of the requirements

Step 2: Show all your unit tests:

```
#include <iostream>
#include "Coordinate.h"  // Include the Coordinate class

void testCoordinate(const char *input, int expectedIndex, bool expectedValid)
{
   Coordinate coord;
   coord.set(input);

   if (coord.getIndex() == expectedIndex && coord.isValid() == expectedValid)
   {
      std::cout << "PASS: '" << input << "' correctly maps to index "
                << expectedIndex << " and valid=" << expectedValid << "\n";
   }

   else {
      std::cout << "FAIL: '" << input << "' mapped to "
                << coord.getIndex() << " (expected " << expectedIndex
                << "), valid=" << coord.isValid()
                << " (expected " << expectedValid << ")\n";
   }
}

int main()
{
   // Valid test cases
   testCoordinate("a1", 0, true);
   testCoordinate("h8", 63, true);
   testCoordinate("C1", 2, true);
   testCoordinate("1C", 2, true);
   testCoordinate("d4", 27, true);
   testCoordinate("8H", 63, true);
   testCoordinate("H8", 63, true);
   testCoordinate("e2", 9, true);

   // Invalid test cases
   testCoordinate("z9", -1, false);  // Out of bounds
   testCoordinate("i5", -1, false);  // Invalid column
   testCoordinate("3x", -1, false);  // Invalid column
   testCoordinate("22", -1, false);  // No valid letter
   testCoordinate("a0", -1, false);  // Invalid row
   testCoordinate("h9", -1, false);  // Invalid row
   testCoordinate("", -1, false);    // Empty input
   testCoordinate("123", -1, false); // Too many characters

   return 0;
}
```

Step 3: Run the test:

```
FAIL: 'a1' mapped to -1 (expected 0), valid=0 (expected 1)
FAIL: 'h8' mapped to -1 (expected 63), valid=0 (expected 1)
FAIL: 'C1' mapped to -1 (expected 2), valid=0 (expected 1)
FAIL: '1C' mapped to -1 (expected 2), valid=0 (expected 1)
FAIL: 'd4' mapped to -1 (expected 27), valid=0 (expected 1)
FAIL: '8H' mapped to -1 (expected 63), valid=0 (expected 1)
FAIL: 'H8' mapped to -1 (expected 63), valid=0 (expected 1)
FAIL: 'e2' mapped to -1 (expected 9), valid=0 (expected 1)
FAIL: 'z9' mapped to -1 (expected -1), valid=0 (expected 0)
FAIL: 'i5' mapped to -1 (expected -1), valid=0 (expected 0)
FAIL: '3x' mapped to -1 (expected -1), valid=0 (expected 0)
FAIL: '22' mapped to -1 (expected -1), valid=0 (expected 0)
FAIL: 'a0' mapped to -1 (expected -1), valid=0 (expected 0)
FAIL: 'h9' mapped to -1 (expected -1), valid=0 (expected 0)
```

```
FAIL: '' mapped to -1 (expected -1), valid=0 (expected 0)
FAIL: '123' mapped to -1 (expected -1), valid=0 (expected 0)
```

Step 4: Write the completed class:

Coordinate.h

```
#ifndef COORDINATE_H
#define COORDINATE_H

class Coordinate
{
private:
   int index;
   bool valid;

public:
   Coordinate() : index(-1), valid(false) {}

   void set(const char *input);
   int getIndex() const { return index; }
   bool isValid() const { return valid; }
};

#endif // COORDINATE_H
```

Coordinate.cpp

```
#include "Coordinate.h"
#include <cctype>  // for std::tolower
#include <cstring> // for strlen

void Coordinate::set(const char *input)
{
   if (input == nullptr || std::strlen(input) < 2 || std::strlen(input) > 3)
   {
      valid = false;
      index = -1;
      return;
   }

   char letter = '\0';
   char number = '\0';

   // Identify the letter (A-H) and number (1-8) in the input
   for (size_t i = 0; input[i] != '\0'; ++i)
   {
      if (std::isalpha(input[i])) letter = std::tolower(input[i]);
      if (std::isdigit(input[i])) number = input[i];
   }

   // Validate the parsed values
   if (letter < 'a' || letter > 'h' || number < '1' || number > '8')
   {
      valid = false;
      index = -1;
      return;
   }

   // Convert to board index (0-63)
   int col = letter - 'a';   // 'a' -> 0, 'h' -> 7
   int row = number - '1';   // '1' -> 0, '8' -> 7

   index = row * 8 + col;    // Compute 0-based index
   valid = true;
}
```

Step 5: Refactor

```
PASS: 'a1' correctly maps to index 0 and valid=1
PASS: 'h8' correctly maps to index 63 and valid=1
PASS: 'C1' correctly maps to index 2 and valid=1
PASS: '1C' correctly maps to index 2 and valid=1
PASS: 'd4' correctly maps to index 27 and valid=1
PASS: '8H' correctly maps to index 63 and valid=1
PASS: 'H8' correctly maps to index 63 and valid=1
PASS: 'e2' correctly maps to index 9 and valid=1
PASS: 'z9' mapped to -1 (expected -1), valid=0 (expected 0)
PASS: 'i5' mapped to -1 (expected -1), valid=0 (expected 0)
PASS: '3x' mapped to -1 (expected -1), valid=0 (expected 0)
PASS: '22' mapped to -1 (expected -1), valid=0 (expected 0)
PASS: 'a0' mapped to -1 (expected -1), valid=0 (expected 0)
PASS: 'h9' mapped to -1 (expected -1), valid=0 (expected 0)
PASS: '' mapped to -1 (expected -1), valid=0 (expected 0)
PASS: '123' mapped to -1 (expected -1), valid=0 (expected 0)
```