**Dr. Dan Bennett**

**Class: RationalT**

**Method Prototype: RationalT()**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Key** | **Testing for** | **Test Case** | **Input/Test value** | **Expected Outcome** | **Observed Result** |
| RT01 | Initializing the variable | RationalT a; | N/A | a.numerator = 0; a.denominator = 1; | As expected |

**Method Prototype: RationalT(int n)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Key** | **Testing for** | **Test Case** | **Input/Test value** | **Expected Outcome** | **Observed Result** |
| RT11 | Initializing the variable | RationalT b(5); | 5 | b.numerator = 5; b.denominator = 1; | As expected |

**Method Prototype: RationalT(int n, int d)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Key** | **Testing for** | **Test Case** | **Input/Test value** | **Expected Outcome** | **Observed Result** |
| RT21 | Initializing the variable | RationalT b(1,2); | Numerator = 1; Denominator = 2 | c.numerator = 5; c.denominator = 1; | As expected |

**Method Prototype: RationalT Add(const RationalT& other) const**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Key** | **Testing for** | **Test Case** | **Input/Test value** | **Expected Outcome** | **Observed Result** |
| A1 | Adding positive whole numbers | Test = d.Add(e) | D = 2; e = 5 | Test = 7 | As expected |
| A2 |  | Test = d.Add(e) | D = 1; e = 4 | Test = 5 | As expected |
| A3 | Adding negative whole numbers | Test = d.Add(e) | D = -5; e = -2 | Test = -7 | As expected |
| A4 |  | Test = d.Add(e) | D = -4; e = -1 | Test = -3 | As expected |
| A5 | Adding positive and negative whole numbers | Test = d.Add(e) | D = -3; e = 0 | Test = -3 | As expected |
| A6 |  | Test = d.Add(e) | D = -2; e = 1 | Test = -1 | As expected |
| A7 | Adding positive fractions | Test = d.Add(e) | D = 1/3; e = 1/5 | Test = 0.53333 | As expected |
| A8 |  | Test = d.Add(e) | D = 2/3; e = 2/5 | Test = 1.06667 | As expected |
| A9 | Adding negative fractions | Test = d.Add(e) | D = -4/3; e = -4/5 | Test = -2.13333 | As expected |
| A10 |  | Test = d.Add(e) | D = -2/3; e = -2/5 | Test = -1.06667 | As expected |

**Method Prototype: RationalT Sub(const RationalT& other) const**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Key** | **Testing for** | **Test Case** | **Input/Test value** | **Expected Outcome** | **Observed Result** |
| S1 | Subtracting positive whole numbers | Test = d.Sub(e) | D = 2; e = 5 | Test = -3 | As expected |
| S2 |  | Test = d.Sub(e) | D = 1; e = 4 | Test = -3 | As expected |
| S3 | Subtracting negative whole numbers | Test = d.Sub(e) | D = -5; e = -2 | Test = -3 | As expected |
| S4 |  | Test = d.Sub(e) | D = -4; e = -1 | Test = -3 | As expected |
| S5 | Subtracting positive and negative whole numbers | Test = d.Sub(e) | D = -3; e = 0 | Test = -3 | As expected |
| S6 |  | Test = d.Sub(e) | D = -2; e = 1 | Test = -3 | As expected |
| S7 | Subtracting positive fractions | Test = d.Sub(e) | D = 1/3; e = 4/5 | Test = -0.46667 | As expected |
| S8 |  | Test = d.Sub(e) | D = 4/3; e = 7/5 | Test = -0.06667 | As expected |
| S9 | Subtracting negative fractions | Test = d.Sub(e) | D = -4/3; e = -1/5 | Test = -1.13333 | As expected |
| S10 |  | Test = d.Sub(e) | D = -2/3; e = -1/5 | Test = -0.86667 | As expected |

**Method Prototype: RationalT Mul(const RationalT& other) const**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Key** | **Testing for** | **Test Case** | **Input/Test value** | **Expected Outcome** | **Observed Result** |
| M1 | Multiplying positive whole numbers | Test = d.Mul(e) | D = 2; e = 5 | Test = 10 | As expected |
| M2 |  | Test = d.Mul(e) | D = 1; e = 4 | Test = 4 | As expected |
| M3 | Multiplying negative whole numbers | Test = d.Mul(e) | D = -5; e = -2 | Test = 10 | As expected |
| M4 |  | Test = d.Mul(e) | D = -4; e = -1 | Test = 4 | As expected |
| M5 | Multiplying positive and negative whole numbers | Test = d.Mul(e) | D = -3; e = 0 | Test = 0 | As expected |
| M6 |  | Test = d.Mul(e) | D = -2; e = 1 | Test = -2 | As expected |
| M7 | Multiplying positive fractions | Test = d.Mul(e) | D = 1/3; e = 4/5 | Test = 0.26667 | As expected |
| M8 |  | Test = d.Mul(e) | D = 4/3; e = 7/5 | Test = 1.86667 | As expected |
| M9 | Multiplying negative fractions | Test = d.Mul(e) | D = -4/3; e = -1/5 | Test = 0.26667 | As expected |
| M10 |  | Test = d.Mul(e) | D = -5/3; e = -2/5 | Test = 0.66667 | As expected |

**Method Prototype: RationalT Div(const RationalT& other) const**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Key** | **Testing for** | **Test Case** | **Input/Test value** | **Expected Outcome** | **Observed Result** |
| D1 | Multiplying positive whole numbers | Test = d.Div(e) | D = 2; e = 5 | Test = 0.4 | As expected |
| D2 |  | Test = d.Div(e) | D = 1; e = 4 | Test = 0.25 | As expected |
| D3 | Multiplying negative whole numbers | Test = d.Div(e) | D = -5; e = -2 | Test = 2.5 | As expected |
| D4 |  | Test = d.Div(e) | D = -4; e = -1 | Test = 4 | As expected |
| D5 | Multiplying positive and negative whole numbers | Test = d.Div(e) | D = -3; e = 0 | ERROR – Division by zero  Test = 0 | As expected |
| D6 |  | Test = d.Div(e) | D = -2; e = 1 | Test = -2 | As expected |
| D7 | Multiplying positive fractions | Test = d.Div(e) | D = 1/3; e = 4/5 | Test = 0.41667 | As expected |
| D8 |  | Test = d.Div(e) | D = 4/3; e = 7/5 | Test = 0.95238 | As expected |
| D9 | Multiplying negative fractions | Test = d.Div(e) | D = -4/3; e = -1/5 | Test = 6.66667 | As expected |
| D10 |  | Test = d.Div(e) | D = -5/3; e = -2/5 | Test = 4.16667 | As expected |

**Method Prototype: int Num(void) const**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Key** | **Testing for** | **Test Case** | **Input/Test value** | **Expected Outcome** | **Observed Result** |
| N1 | Outputting the numerator | a.Num() | A's numerator = 0 | Initialized a has numerator of: 0 | As expected |
| N2 |  | b.Num() | B's numerator = 5 | Initialized b has numerator of: 5 | As expected |
| N3 |  | c.Num() | C's numerator = 1 | Initialized a has numerator of: 1 | As expected |

**Method Prototype: int Denom(void) const**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Key** | **Testing for** | **Test Case** | **Input/Test value** | **Expected Outcome** | **Observed Result** |
| Dn1 | Outputting the denominator | a.Denom() | A's denominator = 1 | and a denominator of: 1 | As expected |
| Dn2 |  | b.Denom() | B's denominator = 1 | and a denominator of: 1 | As expected |
| Dn3 |  | c.Denom() | C's denominator = 2 | and a denominator of: 2 | As expected |

**Method Prototype: double Value(void) const**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Key** | **Testing for** | **Test Case** | **Input/Test value** | **Expected Outcome** | **Observed Result** |
| V1 | Outputting the value | a.Value() | A's value = 0 | and a value of: 0 | As expected |
| V2 |  | b.Value() | A's value = 5 | and a value of: 5 | As expected |
| V3 |  | c.Value() | A's value = 0.5 | and a value of: 0.5 | As expected |

**Method Prototype: void Reduce()**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Key** | **Testing for** | **Test Case** | **Input/Test value** | **Expected Outcome** | **Observed Result** |
| R1 | Properly reducing rational numbers | RationalT f(6,2) | Numerator = 6, denominator = 2 | f.Num() = 3, f.Denom() = 1 | As expected |
| R2 |  | RationalT g(14,10) | Numerator = 14, denominator = 10 | g.Num() = 7, g.Denom() = 5 | As expected |

**Method Prototype: int GCD(int a, int b) const**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Key** | **Testing for** | **Test Case** | **Input/Test value** | **Expected Outcome** | **Observed Result** |
| G1 | Producing the proper GCD | RationalT f(6,2) | Numerator = 6, denominator = 2 | f.Num() = 3, f.Denom() = 1, since the (GCD is used to reduce the fraction to its smallest possible values.) | As expected |
| G2 |  | RationalT g(14,10) | Numerator = 14, denominator = 10 | g.Num() = 7, g.Denom() = 5  (GCD is used to reduce the fraction to its smallest possible values.) | As expected |

**Method Prototype: int LCM(int a, int b) const**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Key** | **Testing for** | **Test Case** | **Input/Test value** | **Expected Outcome** | **Observed Result** |
| L1 | Finding the proper LCM for fractions | Test = f.Add(g) | F = 3, g = 7/5 | Test.Denom() = 5 | As expected |
| L2 |  | Test = g.Add(c) | G = 7/5, c = 1/2 | Test.Denom() = 10 | As expected |