CSE 230 Problem Set 09

Problem 25.1: Compute Pay

Consider the following C++:

double computePay(double hours, double wage)

{

if (hours < 40.0)

return hours \* wage;

else

return (wage \* 40.0) + (wage \* 1.5 \* (hours – 40.0));

}

Create unit tests to exercise the following test cases. Put all the unit tests in a single function.

|  |  |  |
| --- | --- | --- |
| Name | Input (Hour, Wage) | Output |
| Zeros | 0, $0.00 | $0.00 |
| No time | 0, $8.00 | $0.00 |
| One hour | 1, $8.00 | $8.00 |
| No wage | 1, $0.00 | $0.00 |
| Just under full time | 39, $10.00 | $390.00 |
| Full time | 40, $10.00 | $400.00 |
| One hour overtime | 41, $10.00 | $415.00 |
| Double time | 80, $10.00 | $1,000.00 |
| Negative hours | -1, $10.00 | error |
| Negative wage | 1, -$8.00 | error |
| Unreasonable hours | 168, $10.00 | error |

Unit tests:

void test\_computePay()

assert (computePay(0,0.00) == 0.00);

assert (computePay(0, 8.00) == 0.00);

assert (computePay(1, 8.00) == 8.00);

assert (computePay(1,0.00) == 0.00);

assert (computePay(39, 10.00) == 390.00);

assert (comptuePay(40,10.00) == 400.00);

assert (computePay(41, 10.00) == 415.00);

assert (computePay(80,10.00) == 1000.00);

assert (computePay(-1,10.00) == null);

assert(computePay(1,-8.00) == null);

assert (computePay(168, 10.00) == null);)

}

Problem 25.2: Percent

Consider the following class:

class Percent

{

public:

double percent; // stored in value 0.0 – 1.0

Percent() : percent(0.0) {}

double get() { return percent \* 100.0;}

void set(double input)

{

if (input >= 0.0 && input <= 100.0)

percent = input / 100.0;

}

};

Identify test cases for all the methods in the class.

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Setup | exercise | Verify |
| Get Default | myPercent = Percent() | returnValue = myPercent.get() | returnValue = 0.0  myPercent.percent = 0.0 |
| Get 50% | myPercent = Percent()  myPercent.percent = 0.5 | returnValue = myPercent.get() | returnValue = 50.0  myPercent.percent = 0.5 |
| Get 100% | myPercent = Percent()  myPercent.percent = 1.0 | returnValue = myPercent.get() | returnValue = 100.0  myPercent.percent = 1.0 |
| Set 50% | myPercent = Percent() | myPercent.set(50.0) | myPercent.percent = 0.5 |
| Set 100% | myPercent = Percent() | myPercent.set(100.0) | myPercent.percent = 1 |
| Higher than 100% | myPercent = Percent() | myPercent.set(101.0 | Error,  myPercent.percent = 0.0 |
| Lower than 0% | myPercent = Percent() | myPercent.set(-1.0) | Error,  myPercent.percent = 0.0 |

Show two unit tests for the set() method. Make sure that each has the four parts (setup, exercise, verify, teardown). Each unit test should be in its own method in a TestPercent class.

//Get 100%

void TestPercent::???()

{

// setup

Percent myPercent = Percent();

// exercise

Double returnValue = myPercent.get();

// verify

Assert(returnValue = 0.0 && myPercent.percent == 0.0);

// teardown

delete(myPercent);

}

//Set 50%

void TestPercent::???()

{

// setup

Percent myPercent = Percent();

// exercise

myPercent.set(50.0);

// verify

assert(myPercent.percent == 0.5);

// teardown

delete(myPercent);

}

Problem 25.3: Coordinate

Consider the following class diagram designed to represent the position of a piece on a chess board:



Enumerate a set of test cases for each of the public methods:

|  |  |
| --- | --- |
| Method under test | Test Name |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Create a test runner as was done in example 25.3. The class name will be TestCoordinate.

Class TestCoordinate: public TestCase {

Public:

Void run()

{

test\_get\_row\_zero();

test\_get\_row\_one();

test\_get\_row\_twp();

test\_get\_row\_three();

test\_get\_row\_four();

test\_get\_row\_five();

test\_get\_row\_six();

test\_get\_row\_seven();

test\_get\_column\_zero();

test\_get\_column\_one();

test\_get\_column\_twp();

test\_get\_column\_three();

test\_get\_column\_four();

test\_get\_column\_five();

test\_get\_column\_six();

test\_get\_column\_seven();

test\_set\_valid\_coordinates();

test\_set\_invalid\_row();

test\_set\_invalid\_col();

test\_set\_negative\_coordiantes();

test\_set\_too\_large\_coordinates();

test\_display();

}

private:

void test\_get\_row\_zero();

void test\_get\_row\_one();

void test\_get\_row\_twp();

void test\_get\_row\_three();

void test\_get\_row\_four();

void test\_get\_row\_five();

void test\_get\_row\_six();

void test\_get\_row\_seven();

void test\_get\_column\_zero();

void test\_get\_column\_one();

void test\_get\_column\_twp();

void test\_get\_column\_three();

void test\_get\_column\_four();

void test\_get\_column\_five();

void test\_get\_column\_six();

void test\_get\_column\_seven();

void test\_set\_valid\_coordinates();

void test\_set\_invalid\_row();

void test\_set\_invalid\_col();

void test\_set\_negative\_coordiantes();

void test\_set\_too\_large\_coordinates();

void test\_display();

}