Assignment 1 Solution

Your Name and macid

January 11, 2021

This report discusses testing of the `ComplexT` and `TriangleT` classes written for Assignment 1. It also discusses testing of the partner's version of the two classes. The design restrictions for the assignment are critiqued and then various related discussion questions are answered.

1 Assumptions and Exceptions

2 Test Cases and Rationale

3 Results of Testing Partner's Code

4 Critique of Given Design Specification

5 Answers to Questions

(a)

(b) ...

F Code for complex_adt.py

```python
## @file complex_adt.py
#  @author
#  @brief
#  @date

import math

## @brief
#  @details

class ComplexT:

  ## @brief
  #  @details
  #  @param
  #  @param
  def __init__(self, x, y):
    self.x = x
    self.y = y

  ## @brief
  #  @return
  def real(self):
    return self.x

  ## @brief
  #  @return
  def imag(self):
    return self.y

  ## @brief
  #  @return
  def get_r(self):
    return math.sqrt(self.x*self.x + self.y*self.y)

  def get_phi(self):
    return 0.0 #FIXME

  def equals(self, c):
    return (self.x == c.real()) and (self.y == c.imag())

  def conj(self):
    return ComplexT(self.x, self.y) #FIXME

  ## @brief
  #  @return
  def add(self, a):
    return ComplexT(self.x + a.real(), self.y + a.imag())

  def sub(self, a):
    return ComplexT(self.x - a.real(), self.y - a.imag())

  def mult(self, a):
    return ComplexT(self.x - a.real(), self.y - a.imag()) #FIXME

  def recip(self):
    return ComplexT(self.x, self.y) #FIXME

  def div(self, a):
    return ComplexT(self.x - a.real(), self.y - a.imag()) #FIXME

  def sqrt(self):
    return ComplexT(self.x, self.y) #FIXME
```

G Code for triangle_adt.py

```
## @file triangle_adt.py
#  @author
#  @brief
#  @date

from enum import Enum, auto
from itertools import permutations

class TriType(Enum):
    equilat = auto()
    isosceles = auto()
    scalene = auto()
    right = auto()

## @brief
#  @details

class TriangleT:

  ## @brief
  #  @details
  #  @param
  #  @param
  def __init__(self, a, b, c):
    self.s = (a, b, c)

  def get_sides(self):
    return self.s

  def equals(self, t):
    perm = permutations(t.get_sides())
    return (self.s in list(perm))

  def perim(self):
    return 1 #FIXME

  def area(self):
    return 1.0 #FIXME

  def is_valid(self):
    return False #FIXME

  def tri_type(self):
    return TriType.right #FIXME
```

H Code for test_driver.py

```
## @file test_driver.py
#  @author Brooks MacLachlan
#  @brief Tests for date_adt.py and pos_adt.py
#  @date 01/06/2019

import pytest
import copy
from date_adt import DateT
from pos_adt import GPosT

testDate = DateT(14, 6, 1995)
halloween = DateT(31, 10, 2019)
aprilFools = DateT(1, 4, 2019)
newYearsEve = DateT(31, 12, 2019)
newYears = DateT(1, 1, 2020)

testPos = GPosT(50.4, -114.7)
testPos_close = GPosT(50.40001, -114.7)
testPos2 = GPosT(-78.2, 24.0)

# extra test case from Avenue post by Fang Ye
p1 = GPosT(22.59, 113.97)
p2 = GPosT(-34.0522342, -118.2436849)

def test_day():
  expectedDay = 14
  assert testDate.day() == expectedDay

def test_month():
  expectedMonth = 6
  assert testDate.month() == expectedMonth

def test_year():
  expectedYear = 1995
  assert testDate.year() == expectedYear

def test_next():
  expectedTomorrow = DateT(15, 6, 1995)
  assert testDate.next().equal(expectedTomorrow)

def test_next_nextMonth():
  expectedTomorrow = DateT(1, 11, 2019)
  assert halloween.next().equal(expectedTomorrow)

def test_next_nextYear():
  expectedTomorrow = newYears
  assert newYearsEve.next().equal(expectedTomorrow)

def test_prev():
  expectedYesterday = DateT(13, 6, 1995)
  assert testDate.prev().equal(expectedYesterday)

def test_prev_prevMonth():
  expectedYesterday = DateT(31, 3, 2019)
  assert aprilFools.prev().equal(expectedYesterday)

def test_prev_prevYear():
  expectedYesterday = newYearsEve
  assert newYears.prev().equal(expectedYesterday)

def test_before_before():
  assert testDate.before(halloween)

def test_before_after():
  assert not halloween.before(aprilFools)

def test_before_equal():
  assert not testDate.before(testDate)

def test_after_after():
  assert halloween.after(aprilFools)

def test_after_before():
  assert not halloween.after(newYearsEve)

def test_after_equal():
  assert not halloween.after(halloween)

def test_equal_equal():
  assert testDate.equal(testDate)

def test_equal_before():
  assert not halloween.equal(testDate)

def test_equal_after():
  assert not testDate.equal(halloween)

def test_add_days_0():
  assert testDate.add_days(0).equal(testDate)

def test_add_days_5():
  expectedFuture = DateT(19, 6, 1995)
  assert testDate.add_days(5).equal(expectedFuture)

def test_add_days_toNextMonth():
  expectedFuture = DateT(4, 7, 1995)
  assert testDate.add_days(20).equal(expectedFuture)

def test_add_days_toNextYear():
  expectedFuture = DateT(2, 1, 2020)
  assert newYearsEve.add_days(2).equal(expectedFuture)

def test_days_between_before():
  expectedDays = 1
  assert newYears.days_between(newYearsEve) == expectedDays

def test_days_between_after():
  expectedDays = 1
  assert newYearsEve.days_between(newYears) == expectedDays

def test_days_between_equal():
  expectedDays = 0
  assert newYears.days_between(newYears) == expectedDays

def test_GPosT_constructor_latError_low():
  with pytest.raises(Exception) as e:
    GPosT(-178, 0)
  assert "Latitude" in str(e.value)

def test_GPosT_constructor_latError_high():
  with pytest.raises(Exception) as e:
    GPosT(91, 0)
  assert "Latitude" in str(e.value)

def test_GPosT_constructor_longError_low():
  with pytest.raises(Exception) as e:
    GPosT(0, -200)
  assert "Longitude" in str(e.value)

def test_GPosT_constructor_longError_high():
  with pytest.raises(Exception) as e:
    GPosT(0, 181)
  assert "Longitude" in str(e.value)

def test_lat():
  expectedLat = 50.4
  assert testPos.lat() == expectedLat

def test_long():
  expectedLong = -114.7
  assert testPos.long() == expectedLong

def test_west_of_west():
  assert testPos.west_of(testPos2)

def test_west_of_east():
  assert not testPos2.west_of(testPos)

def test_west_of_equal():
  assert not testPos.west_of(testPos)

def test_north_of_north():
  assert testPos.north_of(testPos2)

def test_north_of_south():
  assert not testPos2.north_of(testPos)
```

```
def test_north_of_equal():
  assert not testPos.north_of(testPos)

def test_distance_zero():
  assert testPos.distance(testPos) == 0

def test_distance_nonzero():
  assert testPos.distance(testPos2) == pytest.approx(16510, rel=1e-3)

def test_distance_extra():
  assert p1.distance(p2) == pytest.approx(14804.64385, rel=1e-3)

def test_equal_equalPos():
  assert testPos.equal(testPos)

def test_equal_close():
  assert testPos.equal(testPos_close)

def test_equal_far():
  assert not testPos.equal(testPos2)

def test_move_nowhere():
  testPosCopy = copy.deepcopy(testPos)
  testPosCopy.move(45, 0)
  assert testPosCopy.equal(testPos)

def test_move_somewhere():
  testPosCopy = copy.deepcopy(testPos)
  testPosCopy.move(45, 500)
  assert testPosCopy.lat() == pytest.approx(53, abs=1) and testPosCopy.long() == pytest.approx(-109,
      abs=1)

def test_arrival_date_no_movement():
  assert testPos.arrival_date(testPos, testDate, 1000).equal(testDate)

def test_arrival_date():
  expectedDate = DateT(30, 6, 1995)
  assert testPos.arrival_date(testPos2, testDate, 1000).equal(expectedDate)
```

I Code for Partner's complex_adt.py

```
## @file complex_adt.py
#  @author Partner File
#  @brief
#  @date
```

J Code for Partner's triangle_adt.py

```
## @file triangle_adt.py
#  @author Parnter file
```