Assignment 1 Solution

Aamina Hussain, hussaa54

January 28, 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

For the div method from the ComplexT class, I assumed that the complex numbers created were not of the form z = 0 + 0i, so as to avoid a dividing by zero error. For the sqrt method from the ComplexT class, I assumed that y (where z = x + yi) would not be zero. I raised an exception for the get_phi method of the ComplexT class that returned "The phase is undefined" when the complex number was of the form z = 0 + 0i. Lastly, for the TriangleT class, I assumed that none of the inputs (for any method) would be negative numbers, since side lengths cannot be negative. I also assumed that for the perim and area methods, the triangles would be geometrically valid.

2 Test Cases and Rationale

For most of the methods, I only had one test case. This is because they were simple methods, and as long as the output of the method is the same as the output of the calculator, the method would be correct. The methods I only had one test case for include real, imag, get_r, conj, add, sub, recip, and div from the ComplexT class, and get sides, perim, and area from the TriangleT class. For the rest of the methods, I did the same number of test cases (or more) depending on how many possible outcomes there were for that method. For example, the equal method for both ComplexT and TriangleT have two outcomes: they can either be equal (True) or not equal (False). For this reason, I had two test cases for the equal method (ComplexT). For the TriangleT equal method,

in addition to those two test cases, I had one where the sides were listed in a different order to make sure the method still outputted true regardless of what order the sides were listed in. The is_valid method (TriangleT) also had three test cases since I had to test at least one valid triangle and one invalid triangle. I also only did one test case for sqrt (ComplexT), but afterward, I realized I should have done at lease two, since I had to test when y (where z = x + yi) was positive and when y was negative.

3 Results of Testing Partner's Code

There were no failed test cases when I tested my partner's code using my test_driver.py file. Although we were told it was likely that one or many of our test cases might fail due to ambiguity/abstraction in some parts of the assignment, I was not surprised that my partner's code passed my test cases; taking a closer look at our code, I realized that it was quite similar. I also believe another reason the code passes was because I tried to *generalize* my test cases. However, I was not able to test the tri_type method (TriangleT) on my partner's code. This is because I was unable to finish that specific method for my own assignment, and therefore did not have any test cases for it.

4 Critique of Given Design Specification

The design specification had many strengths, including modularity and abstraction. Modularity because the methods were independent from each other, but were still related. The individual methods could be used in multiple scenarios, not only in the environment they were presented in. Abstraction because the specification only told us what the final result should be for each method, but not how we should implement it. This allowed us more degrees of freedom, and we were able to choose the most efficient way to implement the design.

Some disadvantages includes some unclear instructions. Nothing is specified for what we should do if the integer inputs for TriangleT are negative. It also does not specify what should be done for the methods get_phi, div, and sqrt from ComplexT if the complex number is of the form z = 0 + 0i. What we should do for those cases is not specifically stated in the specification. The design specification was not the most formal, and for this reason there are ambiguities. To improve it I would suggest writing it more formally, or using more formal langauge.

5 Answers to Questions

(a) The real, imag, get_r, get_phi, conj, recip, and sqrt methods from ComplexT and the get_sides, perim, area, is_valid, and tri_type methods from TriangleT are selectors (getters). Assuming that a mutator (setter) is a method that changes or updates the current state of an object, there are no mutator methods. This is because the methods that seem like they are changing the object (such as add from ComplexT) are not actually changing the current object. They are instead returning a newly created object.

(b) State variables for ComplexT could be the real and imaginary parts of the complex number, or the magnitude and phase of the complex number. State variables for TriangleT could be the three sides of the triangle, or the three angles made by the adjacent sides of the triangle.

(c) No, it would not make sense to add methods for less than and greater than. This is because you cannot linearly order complex numbers, since they exist in a two-dimensional plane.

(d) Yes, it is possible that the integer inputs to the constructor for TriangleT will not form a geometrically valid triangle. I believe the class should check if the inputs create a valid triangle before creating the object. If it does not create a valid triangle, an exception should be raised and a suitable message should be outputted. This is provide immediate feedback to the user, so that no errors arise in the future and the performance of the other methods are not affected. Having each method (such as area) check if the triangle is valid before it runs requires an additional step and many compares.

(e) This is a bad idea; the user could input the incorrect type of triangle, which would result in them receiving incorrect information every time they want to access the type. There would be more room for error. Although calculating the type of triangle would take longer than just accessing the state variable, having incorrect data is not desirable. However, if it was guaranteed that the type inputted was correct, then it would be a good idea. Accessing the type of the triangle would be much quicker.

(f) Poor performance usually leads to poor usability. For example, if the execution time of a simple command was more than even a minute, the user would be greatly inconvenienced. This is relevant because usability is the ease at which users can use the product, and it depends strongly on the preferences of the user. If the user is unsatisfied, that means the usability of the software product is insufficient.

(g) It would not be necessary to "fake" a rational design process if whatever you are implementing is simple and straigtforward. However, the "Faked" Rational Design Process is how I usually approach any project, whether it seems simple or not. I figure out the problem, come up with certain requirements that need to be fulfilled in order to solve that problem, then make the code. Then, I test the code to ensure it solves the original problem and meets the requirements. If not, I start again by making some changes.

(h) If a product is reusable, then it means that it can be used to create a new product. In order for the product to be reusable, it should be standardized and generic, meaning it can be used in multiple instances. A product is reliable if it usually does what it is supposed to do. A reusable product is likely to be a reliable product. This is because if it is reusable, then it must be generic, which means it is more likely to work in multiple situations and therefore be more reliable.

(i) Programming languages, especially high-level languages, allow you to write programs that are independent of a specific device or CPU. You can write the program using the high-level languages, such as Java or C++, and then it is converted into machine code using a compiler. Afterwards, it can be run on multiple types of hardware. This abstraction allows for the program to be generic and portable, as well as allowing the programmer more degrees of freedom.

F Code for complex_adt.py

```
## @file complex_adt.py
#  @author Aamina Hussain
#  @brief contains class that has methods for working with complex numbers
#  @date 21/01/2021

from math import sqrt, atan, pi

class ComplexT:

        def __init__(self, x, y):
                self.__x = float(x)
                self.__y = float(y)

        def real(self):
                return self.__x

        def imag(self):
                return self.__y

        def get_r(self):
                return sqrt(self.__x**2 + self.__y**2)

        def get_phi(self):
                if self.__x > 0 or self.__y != 0:
                        return 2*atan(self.__y/(self.get_r()+self.__x))
                elif self.__x < 0 and self.__y == 0:
                        return pi
                else:
                        return "The phase is undefined."

        def equal(self, num):
                if self.__x == num.real() and self.__y == num.imag():
                        return True
                else:
                        return False

        def conj(self):
                return ComplexT(self.__x, -self.__y)

        def add(self, num):
                return ComplexT(self.__x + num.real(), self.__y + num.imag())

        def sub(self, num):
                return ComplexT(self.__x - num.real(), self.__y - num.imag())

        def mult(self, num):
                r = self.__x*num.real() - self.__y*num.imag()
                im = self.__x*num.imag() + self.__y*num.real()
                return ComplexT(r, im)

        def recip(self):
                r = self.__x/(self.__x**2+self.__y**2)
                im = -self.__y/(self.__x**2+self.__y**2)
                return ComplexT(r, im)

        def div(self, num):
                return self.mult(num.recip())

        def sqrt(self):
                r = sqrt((self.__x + self.get_r())/2)
                if self.__y > 0:
                        im = sqrt((-self.__x + self.get_r())/2)
                else:
                        im = -sqrt((-self.__x + self.get_r())/2)
                return ComplexT(r, im)

        #The functions div and sqrt were done assuming that the complex numbers were not of the form z
             = 0 + 0i.
```

G Code for triangle_adt.py

```
## @file triangle_adt.py
#  @author Aamina Hussain
#  @brief contains a class that works with triangles
#  @date 21/01/2021

from math import sqrt, atan, pi
from enum import Enum


class TriangleT:

        def __init__(self, a, b, c):
                self.__a = int(a)
                self.__b = int(b)
                self.__c = int(c)

        def get_sides(self):
                return (self.__a, self.__b, self.__c)

        def equal(self, triangle):
                lstself = [self.__a, self.__b, self.__c]
                lsttri = list(triangle.get_sides())
                lstself.sort()
                lsttri.sort()
                return lstself == lsttri

        def perim(self):
                return int(self.__a + self.__b + self.__c)

        def area(self):
                s = (self.__a + self.__b + self.__c)/2
                A = sqrt(s*(s-self.__a)*(s-self.__b)*(s-self.__c))
                return float(A)

        def is_valid(self):
                if self.__a + self.__b <= self.__c or self.__a + self.__c <= self.__b or self.__b +
                     self.__c <= self.__a:
                        return False
                else:
                        return True
```

H Code for test_driver.py

```
## @file test_driver.py
#  @author Aamina Hussain
#  @brief contains tests for TriangleT (triangle_adt.py) and ComplexT (complex_adt.py) classes
#  @date 21/01/2021

from complex_adt import ComplexT
from triangle_adt import TriangleT


# ***tests for complex_adt.py:

c1 = ComplexT(3,4)
c2 = ComplexT(3,4)
c3 = ComplexT(5,6)
c4 = c1.conj()
c5 = c1.add(c3)
c6 = c1.sub(c3)
c7 = c1.mult(c3)
c9 = c1.div(c3)
c10 = c1.recip()
c11 = c1.sqrt()

# real and imag
if c1.real() == 3 and c2.imag() == 4:
        print("real and imag methods working")
else:
        print("real and imag methods failed")

# get_r
if c1.get_r() == 5:
        print("get_r method working")
else:
        print("get_r method failed")

# equal
if c1.equal(c2) == True and c1.equal(c3) == False:
        print("equal method working")
else:
        print("equal method failed")

# conj
if c4.real() == 3 and c4.imag() == -4:
        print("conj method working")
else:
        print("conj method failed")

# add
if c5.equal(ComplexT(8, 10)):
        print("add method working")
else:
        print("add method failed")

# sub
if c6.real() == -2 and c6.imag() == -2:
        print("sub method working")
else:
        print("sub method failed")

# mult
if c7.real() == -9 and c7.imag() == 38:
        print("mult method working")
else:
        print("mult method failed")

# div
if c9.real() == 0.639344262295082 and c9.imag() == 0.03278688524590162:
        print("div method working")
else:
        print("div method failed")

# recip
if c10.real() == 0.12 and c10.imag() == -0.16:
        print("recip method working")
else:
        print("recip method failed")
```

```
#sqrt
print(c1.sqrt())

################################################################################

# ***tests for triangle_adt.py:

tri1 = TriangleT(3,4,5)
tri2 = TriangleT(3,4,5)
tri3 = TriangleT(7,8,9)
tri4 = TriangleT(4,3,5)
tri5 = TriangleT(1,2,100)

# get_sides
a = tri1.get_sides()
b = tri3.get_sides()
if a == (3,4,5) and b == (7,8,9):
        print("get_sides method working")
else:
        print("get_sides method failed")

# equal
if tri1.equal(tri2) == True and tri1.equal(tri3) == False and tri1.equal(tri4) == True:
        print("equal method working")
else:
        print("equal method failed")

# perim
if tri1.perim() == 12:
        print("perim method working")
else:
        print("perim method failed")

# area
if tri1.area() == 6:
        print("area method working")
else:
        print("area method failed")

# is_valid
if tri1.is_valid() == True and tri3.is_valid() == True and tri5.is_valid() == False:
        print("is_valid method working")
else:
        print("is_valid method failed")
```

I Code for Partner's complex_adt.py

```
## @file complex_adt.py
#  @author Shrill Patel
#  @brief Creates a class that works with complex numbers
#  @date 01/12/21

import math

## @brief Implements an ADT for complex numbers
#  @details A complex number is made up of real and imaginary parts
class ComplexT:

        ## @brief Constructor for the ComplexT class
        #  @details Creates a ComplexT object which creates a complex number
        #                       that has a real and an imaginary part. Assumed the user will
        #                       enter the correct type of input (floats).
        #  @param x Float representing real part of the complex number
        #  @param y Float representing imaginary part of the complex number
        def __init__(self, x, y):
                self.__x = x
                self.__y = y

        ## @brief Gets the real part of the complex number
        #  @return The float representing real portion of the complex number
        def real(self):
                return self.__x

        ## @brief Gets the imaginary part of the complex number
        #  @return The float representing imaginary portion of the complex number
        def imag(self):
                return self.__y

        ## @brief Calculates the magnitude of the complex number
        #  @return The magnitude of the complex number
        def get_r(self):
                return math.sqrt(math.pow(self.real(), 2) + math.pow(self.imag(), 2))

        ## @brief Calculates the argument of the complex number
        #  @details Checks real and imaginary parts of given complex number
        #                       and returns the appropriate angle from positive x-axis in
        #                       radians. This function assumes domain is (-pi, pi).
        #  @return The argument of the complex number
        #  @throws Exception if both real and imaginary parts are 0
        def get_phi(self):
                if self.real() == 0 and self.imag() == 0:
                        raise Exception("Both real and imaginary parts cannot be 0!")
                elif self.real() < 0 and self.imag() == 0:
                        return math.pi
                else:
                        return 2 * math.atan(self.imag() / (self.get_r() + self.real()))

        ## @brief Checks if 2 complex numbers are the same
        #  @details Checks if the real and imaginary parts of 2 complex numbers
        #                       are equal
        #  @param num A complex number of type ComplexT
        #  @return True if both complex numbers are equal, otherwise False
        def equal(self, num):
                return num.real() == self.real() and num.imag() == self.imag()

        ## @brief Calculates the conjugate of a given complex number
        #  @return The complex conjugate of the original complex number
        def conj(self):
                conjugate = self.imag() * -1
                return ComplexT(self.real(), conjugate)

        ## @brief Adds two complex numbers togther
        #  @param complexNum A complex number of type ComplexT
        #  @return The complex number that results from the addition
        def add(self, complexNum):
                return ComplexT(self.real() + complexNum.real(),
                                                self.imag() + complexNum.imag())

        ## @brief Subtracts two complex numbers from each other
        #  @param complexNum A complex number of type ComplexT
        #  @return The complex number that results from the subtraction
        def sub(self, complexNum):
                return ComplexT(self.real() - complexNum.real(),
```

```
                                        self.imag() - complexNum.imag())

    ## @brief Multiplication of two complex numbers
    #  @param num A complex number of type ComplexT
    #  @return The product of two complex numbers
    def mult(self, num):
            return ComplexT(self.real() * num.real() - self.imag() * num.imag(),
                                        self.real() * num.imag() + self.imag() * num.real())

    ## @brief Calculates the reciprocal of a given complex number
    #  @return The reciprocated complex number
    #  @throws Exception if the denominator is 0
    def recip(self):
            denom = math.pow(self.real(), 2) + math.pow(self.imag(), 2)
            if denom == 0:
                    raise Exception("Dividing by 0 is not possible!")
            realPart = self.real() / denom
            imagPart = -1 * (self.imag() / denom)
            return ComplexT(realPart, imagPart)

    ## @brief Divides two complex numbers by each other
    #  @param complexNum A complex number of type ComplexT
    #  @return The quotient of the two complex numbers
    def div(self, complexNum):
            return self.mult(complexNum.recip())

    ## @brief Computes the square root of a complex number
    #  @return The complex number that is produced after applying a
    #                  square root to it
    #  @throws Exception if the imaginary part of the complex number is 0
    def sqrt(self):
            if (self.imag() == 0):
                    raise Exception("Imaginary part of complex number cannot be 0!")
            real = math.sqrt((self.real() + self.get_r()) / 2)
            signum = self.imag() / abs(self.imag())
            imaginary = signum * math.sqrt(((-1 * self.real()) + self.get_r()) / 2)
            return ComplexT(real, imaginary)
```

J Code for Partner's triangle_adt.py

```
## @file triangle_adt.py
#  @author Shrill Patel
#  @brief Creates a class that works with triangles
#  @date 01/14/21

import math
from enum import Enum, auto

## @brief Implements a enumerated class for classifying triangles
#  @details A triangle is equilateral if it has 3 equal sides,
#                          isosceles if it has 2 equal sides, scalene if no sides
#                          are equal, and right if it has a right angle.
class TriType(Enum):
        equilat = auto()
        isosceles = auto()
        scalene = auto()
        right = auto()

## @brief Implements an ADT class for dealing with triangles
#  @details A triangle is made up of three non-negative side lengths
class TriangleT:

        ## @brief Constructor for the TriangleT class
        #  @details Creates a TriangleT object which creates a triangle
        #                          that has non-negative side lengths
        #  @param a Integer representing the first side length of the triangle
        #  @param b Integer representing the second side length of the triangle
        #  @param c Integer representing the third side length of the triangle
        #  @throws Exception if any side length is less than or equal to 0
        def __init__(self, a, b, c):
                if (a <= 0 or b <= 0 or c <= 0):
                        raise Exception("Side length cannot be less than or equal to 0!")
                else:
                        self.__a = a
```

```
                self.__b = b
                self.__c = c

## @brief Fetches all the side length values of given triangle
#  @return The side lengths of the triangle
def get_sides(self):
        return self.__a, self.__b, self.__c

## @brief Checks if all the side lengths of both triangles are equal
#  @param triangle A triangle of type TriangleT
#  @return True if all side lengths are equal, otherwise False
def equal(self, triangle):
        return sorted(triangle.get_sides()) == sorted(self.get_sides())

## @brief Calculates the perimeter of the given triangle
#  @details This function calculates the perimeter of the given
#                   triangle assuming that the triangle is a valid one.
#  @return The sum of all the sides lengths of the triangle
def perim(self):
        sides = self.get_sides()
        return sides[0] + sides[1] + sides[2]

## @brief Calculates the area of the given triangle
#  @details Given a triangle, this function calculates the area of
#                   the given triangle using the Heron's formula (as stated
#                   by https://www.mathsisfun.com/geometry/herons-formula.html).
#                   This function assumes that the input triangle will be a
#                   mathematically valid one.
#  @return The area of the given triangle
def area(self):
        sides = self.get_sides()
        s = self.perim() / 2
        return math.sqrt(s * (s - sides[0]) * (s - sides[1]) * (s - sides[2]))

## @brief Determines if the given side lengths forms a valid triangle
#  @detials Determines if a valid triangle is formed with the given
#                   side lengths. Does so by comparing the sum of two sides
#                   with the third side, if any one of the possible
#                   combinations is less than or equal to the third side, it
#                   is not a valid triangle (as stated by
#                   https://tinyurl.com/y4w465sl).
#  @return True if all sides form valid triangle, false otherwise
def is_valid(self):
        sides = self.get_sides()
        if ((sides[0] + sides[1] <= sides[2]) or
                (sides[0] + sides[2] <= sides[1]) or
                (sides[1] + sides[2] <= sides[0])):
                return False
        else:
                return True

## @brief Determines the type of a triangle
#  @detials Using the enumerated class TriType, this function
#                   classifies triangles into either equilateral,
#                   isosceles, scalene, and/or right. An isosceles
#                   or scalene triangle can also be classified as a
#                   right-angled triangle. In this case, assume
#                   function returns that the triangle is right.
#  @return The TriType corresponding to the correct classification
#                   of a triangle
#  @throws Exception if a valid triangle in not inputed
def tri_type(self):
        if (self.is_valid()):
                s = sorted(self.get_sides())
                if s[0] == s[1] == s[2]:
                        return TriType.equilat
                else:
                        if math.pow(s[0], 2) + math.pow(s[1], 2) == math.pow(s[2], 2):
                                return TriType.right
                        elif s[0] == s[1] or s[1] == s[2] or s[2] == s[0]:
                                return TriType.isosceles
                        else:
                                return TriType.scalene
        else:
                raise Exception("Invalid triangle given!")
```

K Citations

1. Wikipedia: Complex Numbers
2. Wikipedia: Hardware Abstraction
3. SE 2AA4 lecture slides