Assignment 1 Solution

Shazil Arif, 400201970

January 14, 2020

Intro blurb.

1 Testing of the Original Program

Description of approach to testing. Rationale for test case selection. Summary of results. Any problems uncovered through testing.

2 Results of Testing Partner's Code

Consequences of running partner's code. Success, or lack of success, running test cases. Explanation of why it worked, or didn't.

3 Critique of Given Design Specification

Advantages and disadvantages of the given design specification.

4 Answers to Questions

(a)

E Code for date_adt.py

```
## @file date_adt.py
#  @author Shazil Arif
#  @brief date_adt.py contains a Class that implements a Date object containing a year, month and day
#  @date Jan 8th, 2020

from datetime import datetime
from datetime import timedelta

## @brief DateT is a class that implements a Date object containing a year, month and a day
class DateT:

  ## Represents the months with 31 days
  odd_month = [1,3,5,7,8,10,12]

  ## enum for representing the maximum number of days in the months contained in odd_month
  max_odd_month = 31

  ## Represents months with 30 days
  even_month = [4,6,9,11]

  ## enum for representing the maximum number of days in the months contained in even_month
  max_even_month = 30

  ## enum for the 12 month
  december = 12

  ## enum for the month of february
  february = 2

  ## enum for the first month
  january = 1

  ## enum for number of days in a leap year in the month of february
  leap_year_days = 29

  ## enum for number of days in february in a common year (not a leap year)
  feb_common_days = 28

  ## @brief the constructor method for class DateT
  #  @param d The date to be set
  #  @param m the Month to be set
  #  @param y the Year to be set
  def __init__(self, d, m, y):
    self.d = d
    self.m = m
    self.y = y

  ## @brief returns the day
  #  @return the day
  def day(self):
    return self.d

  ## @brief returns the month
  #  @return the month
  def month(self):
    return self.m

  ## @brief returns the year
  #  @return the year
  def year(self):
    return self.y

  ## @brief returns a DateT object that is 1 day later than the current object
  #  @return DateT object that is set 1 day later
  def next(self):
    #going into new month when current month has 31 days
    if(self.month() in self.odd_month and self.day() + 1 > self.max_odd_month and self.month() !=
        self.december):
      return DateT(1  , self.month() + 1, self.year())

    #going into new month when current month has 30 days
    if(self.month() in self.even_month and self.day() + 1 > self.max_even_month):
      return DateT(1, self.month() + 1, self.year())

    #going into the new year
    if(self.day()+ 1 > self.max_odd_month and self.month() == self.december):
```

```
    return DateT(1, 1, self.year() + 1)

  # if current month is february
  if(self.month() == self.february):
    #leap year and transitioning into march
    if(self.__is_leap_year() and self.day() + 1 > self.leap_year_days):
      return DateT(1, self.month() + 1, self.year() )

    #not a leap year but transitioning into march
    elif(not self.__is_leap_year() and self.day() + 1 > self.feb_common_days):
      return DateT(1, self.month() + 1, self.year())

  #otherwise return the next day in the current month and year
  return DateT(self.day() + 1, self.month(), self.year())

## @brief returns a DateT object that is 1 day before the current object
#  @return DateT object that is set 1 day before
def prev(self):
  #in the case where we go back to the previous month
  if(self.day() - 1 < 1 and self.month() != self.january):

    #if previous month is not february
    if(self.month() - 1 != self.february):
      #check if previous month has 31 days
      if(self.month() - 1 in self.odd_month):
        return DateT(self.max_odd_month, self.month() - 1, self.year())

      #previous month has 30 days
      if(self.month() - 1 in self.even_month):
        return DateT(self.max_even_month, self.month() - 1, self.year())

    #in the case where previous month is february
    #first check if leap year or not
    if(self.__is_leap_year()):
      return DateT(self.leap_year_days,self.february,self.year())
    return DateT(self.feb_common_days, self.february, self.year())


  #in the case we have to go back to the previous year
  if(self.day() - 1 < 1 and self.month() == self.january):
    return DateT(self.max_odd_month, self.december, self.year() - 1)

  #the simplest case, where there is no month or year transition
  return DateT(self.day() - 1, self.month(), self.year())


## @brief compares if the date represented by the current DateT object is before d (d is also a
    DateT object)
#  @param d The DateT object to compare with the current object
#  @return A boolean value indicating whether the current objects date is before the date in d (True
    if before, False otherwise)
def before(self,d):
  if(self.year() < d.year()): return True
  if(self.year() == d.year() and self.month() < d.month()): return True
  if(self.year() == d.year() and self.month() == d.month() and self.day() < d.day()): return True
  return False

## @brief compares if the date represented by the current DateT object is after d (d is also a DateT
    object)
#  @param d The DateT object to compare with the current object
#  @return A boolean value indicating whether the current objects date is after the date in d (True
    if before, False otherwise)
def after(self,d):
  if (not self.before(d)): return True
  return False

## @brief compares if the current DateT object and another DateT object d represent the same date
#  @param d The DateT object to compare with the current object
#  @return A boolean value indicating whether the two objects represent the same data (True if
    equal, False otherwise)
def equal(self,d):
  return self.__dict__ == d.__dict__

## @brief adds n days to the date represented by the current DateT object
#  @param n The number of days to add
#  @return A DateT object with its date set n days later than the original
def add_days(self,n):
  temp = datetime(self.year(), self.month(), self.day())
  temp = temp + timedelta(days=n)
  return DateT(temp.day,temp.month,temp.year)
```

```
## @brief calculates the number of days between the current DateT object and DateT object d
#  @param d The DateT object to calculate the number of days in between with
#  @return An integer value indicating the number of days between the two DateT objects
def days_between(self,d):
  date_one = datetime(self.year(), self.month(), self.day())
  date_two = datetime(d.year(),d.month(),d.day())
  difference = date_one - date_two
  return abs(difference.days)

## @brief returns whether or not the year in the current DateT object is a leap year
#  @return a boolean value indicating whether or not the year is a leap year (True if leap year,
    False otherwise)
def __is_leap_year(self):
  if(self.year() % 400 == 0): return True
  if(self.year() % 100 == 0): return False
  if(self.year() % 4 == 0): return True
  return False
```

F Code for pos_adt.py

```
## @file pos_adt.py
#  @author Shazil Arif
#  @brief pos_adt.py implements a class for global position coordinates
#  @date January 9th, 2020

import math as Math
import date_adt as Date
from pytest import *
## @brief GPosT is class that implements an object to represent coordinates using longitude and
    latitude values
class GPosT:
    ## @brief the constructor method for class GPost
    #  @param phi The latitude to be set for the GPost object
    #  @param _lambda the longitude value to be set for the GPost object
    def __init__(self, phi, _lambda):
        self.latitude = phi
        self.longitude = _lambda


    ## @brief returns the latitude for the current GPost object
    #  @return the latitude value
    def lat(self):
        return self.latitude

    ## @brief returns the longitude for the current GPost object
    #  @return the longitude value
    def long(self):
        return self.longitude

    ## @brief returns whether the coordinates of the current GPost object are west of those in object p
    #  @param p the GPost object to compare
    #  @return a boolean value indicating whether the current objects coordinates are west of p (True
        if they are west of p, False otherwise)
    def west_of(self,p):
        return self.long() < p.long()

    ## @brief returns whether the coordinates of the current GPost object are north of those in object
        p
    #  @param p the GPost object to compare
    #  @return a boolean value indicating whether the current objects coordinates are north of
        coordinates in p (True if they are west of p, False otherwise)
    def north_of(self,p):
        return self.lat() > p.lat()

    ## @brief returns whether the current GPost object and a GPost object p represent the same position
    #  @details considered to represent the same location if the distance between their coordinates is
        less than 1 km
    #  @param p the GPost object to compare againt
    #  @return a boolean value indicating whether the two objects represent same location(i.e if their
        distance is less than 1km). True if same location, False otherwise
    def equal(self,p):
        return self.distance(p) < 1


    ## @brief moves the position represented by the current GPost object in direction of bearing b
        with total distance d
    #  @param b A real number indicating the bearing/direction to move in
    #  @param d A real number indicating the distance to move in units of kilometres (km)
    def move(self,b,d):
        radius = 6371

        phi_one = Math.radians(self.lat())
        angular_dist = d/radius

        new_lat = Math.asin(Math.sin(phi_one) * Math.cos(angular_dist) +
            Math.cos(phi_one)*Math.sin(angular_dist)*Math.cos(Math.radians(b)) )
        new_long = self.long() +
            Math.degrees(Math.atan2(Math.sin(Math.radians(b))*Math.sin(angular_dist)*Math.cos(phi_one),
            Math.cos(angular_dist) - Math.sin(phi_one) * Math.sin(new_lat) ))

        self.latitude = Math.degrees(new_lat)
        self.longitude = (new_long)

    ## @brief calculates the distance between the positions represented by current GPost object and
        another GPost object 'p'
```

```
#  details Applies the spherical law of cosines formula to calculate the distance. See
     https://www.movable-type.co.uk/scripts/latlong.html under the heading 'Spherical Law of
     Cosines'
#  @param p A GPost object containing the lat/long coordinates to calculate the distance to
#  @return an integer value representing the distance between the current object and p in units of
     kilometres(km)
def distance(self,p):
    #earth's approximate radius in kilometres
    radius = 6371

    lat_one = Math.radians(self.lat())
    lat_two = Math.radians(p.lat())

    long_diff = Math.radians(p.long() - self.long())

    distance = Math.acos(Math.sin(lat_one)*Math.sin(lat_two) +
        Math.cos(lat_one)*Math.cos(lat_two)*Math.cos(long_diff)) * radius

    return distance

## @brief calculates the number of days required to travel from the position represented by
     current GPost object to another position represented by a GPost object while travelling at a
     specific speed and starting on a specific day
#  @param p A GPost object representing the position to travel to
#  @param d a DateT object respresenting the date to begin travelling on
#  @param s A real number indicating the speed to travel at in units of km/day
#  @return an integer value representing the distance between the current object and p in units of
     kilometres(km)
def arrival_date(self,p,d,s):
    distance = self.distance(p)

    #number of days required to cover the distance travelling at speed s
    num_days = Math.ceil(distance/s)

    return d.add_days(num_days)
```

G Code for test_driver.py

```
## @file test_driver.py
#  @author Shazil Arif
#  @brief this test driver module is used to test modules DateT and GPost
#  @date January 10th, 2020
from date_adt import DateT
from pos_adt import GPosT

failed = []

def compare(description, expected, actual):
    print("Description: {description}\n".format(description=description))

    #if expected value is instance of DateT or GPost class
    if(isinstance(expected,DateT) or isinstance(expected,GPosT)):
        expected_keys = expected.__dict__
        actual_keys = actual.__dict__
        print("Expected properties")
        for i in expected_keys:
            print("{key} : {value}".format(key=i,value=expected_keys[i]))
        print("\nActual properties")
        for i in actual_keys:
            print("{key} : {value}".format(key=i,value=actual_keys[i]))
        if(expected.__dict__ == actual.__dict__):
            print('\nResult: ' + '\x1b[6;30;42m' + 'Passed' + '\x1b[0m') #source:
                https://stackoverflow.com/questions/287871/how-to-print-colored-text-in-terminal-in-python
        else:
            failed.append({"Description":description,"Expected":expected,"Actual":actual})
            print('\nResult: ' + '\x1b[1;37;41m' + 'Failed' + '\x1b[0m')

    else: #comparing other types...string, int, float etc.
        print("Expected: {expected}".format(expected=expected))
        print("Actual:   {actual}".format(actual=actual))
        if(expected == actual): print('\nResult: ' + '\x1b[6;30;42m' + 'Passed' + '\x1b[0m')
        else:
            failed.append({"Description":description,"Expected":expected,"Actual":actual})
            print('\nResult: ' + '\x1b[1;37;41m' + 'Failed' + '\x1b[0m')
    print("\n----------------------------------------------------\n")

def test_date_adt():
    #testing date_adt.py

    #2020 is a leap year!
    test = DateT(1,1,2020)

    #test constructor
    compare("test for constructor",1,test.d)
    compare("test for constructor",1,test.m)
    compare("test for constructor",2020,test.y)

    #test getter
    compare("testing getter method for day",1, test.day())
    compare("testing getter method for month",1, test.month())
    compare("testing getter method for year",2020, test.year())


    #test next function
    #ideally for a functions like this the number of tests to run should be equal to or greater than
        the number of execution paths
    #there are 6 cases
    # i) simply the next day within current month and year
    # ii) Transition into The next month where the current month has 30 days
    # iii) Transition into The next month where the current month has 31 days
    # iv) Transition into The next year
    # v) Transition into the next month when the current month is february and it a leap year
    # vi) Transition into the next month when current month is february and it is not a leap year

    compare("testing next method, it should return January 2nd 2020 and
        pass",test.next(),DateT(2,1,2020))

    test = DateT(31,1,2020)
    compare("test for transitioning into next month with current month having 31 days. It should
        return february 1st 2020 and pass",test.next(),DateT(1,2,2020))

    test = DateT(30,4,2020) #April 30th, 2020
    compare("test for transitioning into next month with current month having 30 days. It should
        return May 1st 2020 and pass",test.next(),DateT(1,5,2020))
```

```
test = DateT(28,2,2020)
compare("test for transitioning into next month with current month being february and the year is
    a leap year. It should return Feb 29th 2020 and pass",test.next(),DateT(29,2,2020))

test = DateT(28,2,2021)
compare("test for transitioning into next month with current month being february and the year is
    NOT leap year. It should return March 1st 2021 and pass",test.next(),DateT(1,3,2021))

test = DateT(31,12,2020)
compare("test for transitioning into next year. It should return Jan 1st 2021 and
    pass",test.next(),DateT(1,1,2021))

#test prev method
test=DateT(2,1,2020)
compare("test for prev method, it should return January 1st 2020 and
    pass",DateT(1,1,2020),test.prev())

test=DateT(1,5,2020)
compare("test for transitioning into previous month with current month having 31 days. It should
    return April 30th 2020 and pass",DateT(30,4,2020),test.prev())

test=DateT(1,6,2020)
compare("test for transitioning into previous month with current month having 30 days. It should
    return May 31st 2020 and pass",DateT(31,5,2020),test.prev())

test=DateT(1,3,2020)
compare("test for transitioning back into february and the year is a leap year. It should return
    Feb 29th 2020 and pass",DateT(29,2,2020),test.prev())

test=DateT(1,3,2021)
compare("test for transitioning back into february and the year is NOT leap year. It should return
    Feb 28th 2021 and pass",DateT(28,2,2021),test.prev())

test=DateT(1,1,2020)
compare("test for transitioning into previous year. It should return Dec 31st 2019 and
    pass",DateT(31,12,2019),test.prev())

#test for before method
test  = DateT(1,1,2020)
test2 = DateT(1,5,2020)

compare("test for before method , it should return True and pass",True,test.before(test2))
compare("test for before method , it should return False and pass",False,test2.before(test))

#test for after method
compare("test for after method , it should return True and pass",True,test2.after(test))
compare("test for after method , it should return False and pass",False,test.after(test2))

#test equals method
test = DateT(1,1,2020)
test2 = DateT(1,1,2020)
test3 = DateT(1,2,2020)
compare("test for equals method, it should return True and pass",True,test.equal(test2))
compare("test for equals method, it should return False and pass",False,test.equal(test3))

#test add_days method
test = DateT(31,1,2020)
compare("test add days method, it should return Feb 1st 2020 and
    pass",DateT(1,2,2020),test.add_days(1))
compare("test add days method, it should return Feb 29, 2020", DateT(29,2,2020),test.add_days(29))

test = DateT(31,1,2021)
compare("test add days method, it should return March 1st, 2021",
    DateT(1,3,2021),test.add_days(29))

test = DateT(1,1,2021)
compare("test add days method, add 365 days when current year is NOT leap year, it should return
    january 1st 2022",DateT(1,1,2022),test.add_days(365))

test = DateT(1,1,2020)
compare("test add days method, add 365 days when current year IS LEAP YEAR. it should return Dec
    31st, 2020",DateT(31,12,2020),test.add_days(365))

test = DateT(1,1,2020)
compare("test add days method, add 366 days when current year IS LEAP YEAR. it should return Jan
    1st, 2021",DateT(1,1,2021),test.add_days(366))


#test days_between method
```

```
    test = DateT(31,1,2020)
    test2 = DateT(1,3,2020)

    compare("test days_between method with March and January when current year is leap year, it should
        return 30 days",30,test2.days_between(test))

    test = DateT(31,1,2021)
    test2 = DateT(1,3,2021)
    compare("test days_between method with March and January when current year is NOT leap year, it
        should return 29 days",29,test2.days_between(test))



def test_post_adt():
    test = GPosT(45,45)
    compare("test for constructor",45, test.latitude)
    compare("test for constructor",45, test.longitude)

    compare("test getter method for latitude",45,test.lat())
    compare("test getter method for longitude",45,test.long())

    #compare("test west_of method",)

    test = GPosT(43.580605, -79.625668)
    test2 = GPosT(40.723606, -73.860514)
    compare("test distance method , it should return 571km rounded to the nearest whole
        number",571,int(test2.distance(test)))

    test = GPosT(43.261897, -79.921433)
    test2 = GPosT(43.262545, -79.922549)
    compare("test equal method for distance < 1 km, it should return True",True, test2.equal(test))

    test2 = GPosT(43.250880, -79.920292)
    compare("test equal method for distance > 1km, it should return False",False,test2.equal(test))

    test = GPosT(45,45)
    test2 = GPosT(45,-45)
    compare("test west_of method , it should return True",True,test2.west_of(test))

    compare("test west_of method, it should return False",False,test.west_of(test2))

    test = GPosT(45,45)
    test2 = GPosT(50,-45)
    compare("test north_of method, it should return True",True,test2.north_of(test))
    compare("test north_of method, it should return False",False,test.north_of(test2))

    test = GPosT(43,-75)
    test2 = GPosT(44.078061, -73.170068)
    test.move(45,100)
    compare("test move method",GPosT(44.078061, -73.170068),test)



def main():
    print("Tests for date_adt.py")
    test_date_adt()

    print("\nTESTS FOR pos_adt.py")
    test_post_adt()

    if(len(failed)!=0):
        print("\x1b[1;37;41m {num} tests failed. The following tests failed: \x1b[0m
            \n".format(num=len(failed)))
        for i in range(len(failed)):
            print("{num} )
                Description:{Description}".format(num=(i+1),Description=failed[i]["Description"]))
    else: print("\x1b[6;30;42m All tests passed \x1b[0m")


main()
```

H Code for Partner's CalcModule.py

```
## @file pos_adt.py
#  @author Partner
```