Assignment 2 Solution

Yujing Chen, cheny48

February 19, 2019

Report for Assignment 2

1 Testing of the Original Program

At least one case is used for each function. pytest is used for testing. For each function, it has at least one test for normal case and one test for exception. For some function, there are tests towards the boundary case. For example, in allocate(), besides exception case, case that one department is full is also tested. Initializing is also tested to see if the set is correctly initialized. My program pass 31/31 of the test. In the first round of testing, SALst.average() has failed because of the error comes from double comparing in python.

2 Results of Testing Partner's Code

My partner's code passed 27 out of 31 test cases. This is because he used List structure and i used Set structure in DCapALst and SALst, so the constructor testing and the testing involve finding a certain element in the list fail, as i used String as key in my test and he need integers or slices. Except this, all the other function pass the test.

3 Discussion of the test results

My program didn't use NamedTuple, and my partner didn't use Set structure in DCapALst and SALst. However, when it come to test, both our code still can pass most of the testing case, except the one need specific data structure. So when writing a test file for code, we should pay attention to those involve data structure, and if possible, avoid it to make the test file more flexible.

4 Critique of Given Design Specification

Using specification to represent basically everything could make the code more consistent, it also makes the code easier to modify in the future. I like the idea that separate program into DCap list, Student list and Associated list. This make the program clearer. But as we broke the code into so many module, the probability of error also increases. I doesn't like the design that GenT and DeptT is initialized in StdntAllocTypes. Because we already have data set, and those type is repeated. It may be better if we keep those type empty in StdntAllocTypes, and initialize it later in the program. In that way, those two types could become more flexible. Also, this specification makes many assumption about the input. Those assumption should also be implemented into the program as checking.

5 Answers

1. The formal instructions is more accurate, more efficient if the expected reader can understand it. It also make the code more consistent, it also makes the code easier to modify in the future. However, it can also be very confusing if the programmer doesn't understand it. The natural language is easier to be understood, and it provide programmer more flexibility to implement the program. But it is less convenient to be documented, and could cause misunderstanding between the client and the programmer.

2. We can add a gpa checking in the function get-gpa(). If the gpa is not between 0 and 12, we might throw an exception. In this case, we can keep all the old type in our specification and do not need to add a new ADT.

3. We could document the similar function of SALst and DCapLst together in another part in the specification Then in SALst and DCapLst own part we extend from the similar function part, so we only need to document their own specific function, and could avoid duplicate of the similar function.

4. A2 uses ADT instead of predefined data structures, therefore, it can run with different data structure as long as you use the same ADT. In this case, A2 is more general than A1.

5. Using SeqADT allowing we use same function on different data structure as long as the structure is developed from SeqADT. Therefore, it would be easier to manage the code. Also, using SeqADT would improves generality, it makes reusing of the code in other modules more possible.

6. Using enums makes access in the keys and values easier, and as enums use integer as key would allow operation like addition and multiplication on it, and also enums make iteration of the list easier. However, macid should not use enmus, as there are too many different macid, and those macid is inputed from the file and we don't know what are them in advance, also we would not know their index in advance. So using enmus would not help as find a certain macid, but would make it more difficult for us.

F Code for StdntAllocTypes.py

```
## @file StdntAllocTypes.py
#  @author Yujing Chen
#  @brief recording needed data and data type
#  @date 02/11/2019
from SeqADT import *


## @brief data for Gender
class GenT:

    male = "male"
    female = "female"


## @brief data for DeptT
class DeptT:

    civil = "civil"
    chemical = "chemical"
    electrical = "electrical"
    mechanical = "mechanical"
    software = "software"
    materials = "materials"
    engphys = "engphys"


## @brief structure type of SInfoT
class SInfoT:

    def __init__(self, fname, lname, gender, gpa, choices, freechoice):
        self.fname = fname
        self.lname = lname
        self.gender = gender
        self.gpa = float(gpa)
        self.choices = choices
        self.freechoice = freechoice
```

G Code for SeqADT.py

```
## @file SeqADT.py
#  @author Yujing Chen
#  @brief General function for SeqADT
#  @date 2019/02/11


## @brief An abstract data type that represents a Sequence
class SeqADT:
    ## @brief SeqADT constructor
    #  @details initilize a SeqADT for a sequence, s for sequence, i for index
    #  @param x sequence of T
    def __init__(self, x):
        self.i = 0
        self.s = x

    ## @brief start change the index to start of Seq
    #  @return N/A
    def start(self):
        self.i = 0

    ## @brief next return the current value and add 1 to index
    #  @return value of s[i]
    def next(self):
        if self.i >= len(self.s):
            raise StopIteration
        out = self.s[self.i]
        self.i += 1
        return out

    ## @brief end returns whether the Seq has end
    #  @return boolean, True if Seq end, false if not
    def end(self):
        return self.i >= len(self.s)
```

H Code for DCapALst.py

```
## @file DCapALst.py
#  @author Yujing Chen
#  @brief create the DCapLst for the program
#  @date 02/11/2019
from StdntAllocTypes import *


class DCapALst:

    s = {}
    ## @brief intialize a DCapLst
    #  @param N/A
    #  @return N/A
    @staticmethod
    def init():
        DCapALst.s = {}

    ## @brief add department to the DCapLst
    #  @param d string of department
    #  @param n capacity number
    #  @return N/A
    #  @exception KeyError
    @staticmethod
    def add(d, n):
        for dep in DCapALst.s:
            if dep == d:
                raise KeyError
        DCapALst.s[d] = n

    ## @brief remove department from the DCapLst
    #  @param d string of department
    #  @return N/A
    #  @exception KeyError
    @staticmethod
    def remove(d):
        if d in DCapALst.s:
            del DCapALst.s[d]
        else:
            raise KeyError

    ## @brief define if an element is in the list
    #  @param d string of department
    #  @return True if element in the list, false if not
    @staticmethod
    def elm(d):
        for dep in DCapALst.s:
            if dep == d:
                return True
            else:
                return False

    ## @brief return the capacity of a department of the DCapLst
    #  @param d string of department
    #  @return the capacity of department d
    #  @exception KeyError
    @staticmethod
    def capacity(d):
        for dep in DCapALst.s:
            if dep == d:
                return DCapALst.s[dep]
        raise KeyError
```

I Code for AALst.py

```
## @file AALst.py
#  @author Yujing Chen
#  @brief create the AALst for the program
#  @date 02/11/2019
import StdntAllocTypes


class AALst:

    s = {}

    ## @brief initialize the AALst
    #  @param N/A
    #  @details create the AALst in the form of {deparment: []}
    #  @return N/A
    @staticmethod
    def init():
        AALst.s = {}
        AALst.s["civil"] = []
        AALst.s["materials"] = []
        AALst.s["chemical"] = []
        AALst.s["electrical"] = []
        AALst.s["mechanical"] = []
        AALst.s["software"] = []
        AALst.s["engphys"] = []

    ## @brief add student to the AALst list
    #  @param dep department
    #  @param m student id
    #  @return N/A
    @staticmethod
    def add_stdnt(dep, m):
        if dep in AALst.s:
            AALst.s[dep].append(m)

    ## @brief give the student list of a department
    #  @param d string of department
    #  @return return the student list that assign to a department
    @staticmethod
    def lst_alloc(dep):
        return AALst.s[dep]

    ## @brief give the student list length of a department
    #  @param d string of department
    #  @return the length of the student list
    @staticmethod
    def num_alloc(dep):
        return len(AALst.s[dep])
```

J Code for SALst.py

```
## @file SALst.py
#  @author Yujing Chen
#  @brief create the SALst for the program
#  @date 02/11/2019
from AALst import *
from StdntAllocTypes import *
from DCapALst import *


class SALst:

    s = {}

    ## @brief initialize of SALst
    #  @return N/A
    @staticmethod
    def init():
        SALst.s = {}

    ## @brief add student information to SALst
    #  @param m the student id of the student
    #  @param i the SInfo of the student
    #  @return N/A
    #  @exception KeyError when element already in the list
    @staticmethod
    def add(m, i):
        for id in SALst.s:
            if id == m:
                raise KeyError
        SALst.s[m] = i

    ## @brief remove the student from the SALst
    #  @param m the student id of the student
    #  @return N/A
    #  @exception KeyError when element not in the list
    @staticmethod
    def remove(m):
        if m not in SALst.s:
            raise KeyError
        else:
            del SALst.s[m]

    ## @brief find if a student is in the SALst
    #  @param m the student id of the student
    #  @return true if the element in the list, false if not
    @staticmethod
    def elm(m):
        for id in SALst.s:
            if id == m:
                return True
            else:
                return False

    ## @brief return the student infomation of m
    #  @param m the student id of the student
    #  @return the coresponding student information
    #  @exception KeyError when element is not in the list
    @staticmethod
    def info(m):
        if m not in SALst.s:
            raise KeyError
        else:
            return SALst.s[m]

    ## @brief function to return student's gpa
    #  @param m the student id of m
    #  @return the gpa of m
    @staticmethod
    def get_gpa(m):
        return SALst.s[m].gpa

    ## @brief sorting student according to their gpa
    #  @param f the function that the student info need to satisfy
    #  @return the sorted list
    @staticmethod
    def sort(f):
```

```
        temp = []
        for i in SALst.s:
            if f(SALst.s[i]):
                gpa = SALst.get_gpa(i)
                temp.append((i, gpa))
        temp.sort(key=lambda x: x[1], reverse=True)
        sorted = []
        for i in temp:
            sorted.append(i[0])
        return sorted

    ## @brief calculate the average of a group of student
    #  @param f the function that a student info must satisfy
    #  @return average
    #  @exception ValueError when the list is empty
    @staticmethod
    def average(f):
        temp = []
        for i in SALst.s:
            if f(SALst.s[i]):
                gpa = SALst.get_gpa(i)
                temp.append((i, gpa))
        if len(temp) == 0:
            raise ValueError
        average = sum([i[1] for i in temp]) / len(temp)
        return average

    ## @brief allocate student to coresponding AALst
    #  @param N/A
    #  @return N/A
    #  @exception RuntimeError when a student can't be allocate
    @staticmethod
    def allocate():
        F = SALst.sort(lambda t: t.freechoice and t.gpa >= 4.0)
        for m in F:
            ch = SALst.s[m].choices
            AALst.add_stdnt(ch.next(), m)
        S = SALst.sort(lambda t: t.freechoice is False and t.gpa >= 4.0)
        for m in S:
            ch = SALst.s[m].choices
            alloc = False
            while alloc is False and ch.end() is False:
                d = ch.next()
                if AALst.num_alloc(d) < DCapALst.capacity(d):
                    AALst.add_stdnt(d, m)
                    alloc = True
            if alloc is False:
                raise RuntimeError
```

K Code for Read.py

```
## @file Read.py
#  @author Yujing Chen
#  @brief read the input data from serveral file
#  @date 02/11/2019
import StdntAllocTypes
from DCapALst import *
from SALst import *

import ast


## @brief read the student information from a file, and store it into SALst
#  @param s a string s corresponding to a filename
#  @return N/A
def load_stdnt_data(s):
    SALst.init()
    with open(s) as f:
        for line in f:
            (id, fname, lname, gender, gpa, choices,
             freechoice) = line.split(', ')
            choices = ast.literal_eval(choices)
            freechoice = ast.literal_eval(freechoice)
            student = SInfoT(fname, lname, gender, gpa,
                             SeqADT(choices), freechoice)
            SALst.add(id, student)


## @brief read the department depth information from a file
#  and add it into the DcapAlst
#  @param s a string s corresponding to a filename
#  @return N/A.
def load_dcap_data(s):
    DCapALst.init()
    with open(s) as f:
        for line in f:
            (key, val) = line.split(', ')
            DCapALst.add(key, int(val))
```

L Code for Partner's SeqADT.py

```
## @file SeqADT.py
# @author Seva Skvortsov
# @brief SeqADT
# @date 08/02/2019


## @brief An abstract date type to storing a sequence
class SeqADT:
    ## @brief init initial data structure
    #  @param x the sequence to store
    def __init__(self, x):
        self.s = x
        self.i = 0

    ## @brief returns the index of sequence to 0 (start)
    def start(self):
        self.i = 0

    ## @brief gives current element moves index to next element
    #  @exception throws StopIteration if the next index is out of sequence
    #  @return the element at the current index
    def next(self):
        if self.i >= len(self.s):
            raise StopIteration
        temp = self.i
        self.i += 1
        return self.s[temp]

    ## @brief tells you if you are at the end of the sequence
    #  @return True if index greater or equal to than length of the sequence False else
    def end(self):
        return (self.i >= len(self.s))
```

M Code for Partner's DCapALst.py

```
## @file DCapALst.py
# @author Seva Skvortsov
# @brief DCapALst
# @date 08/02/2019

from StdntAllocTypes import *
from typing import NamedTuple


## @brief abstract object for Department Capacity Association
class DCapALst:
    ## @brief local tuple class to represent tuple of department and its capacity
    class __Tuple__(NamedTuple):
        dept: DeptT
        cap: int
    ## @brief initializes the abstract object
    @staticmethod
    def init():
        DCapALst.s = []
    ## @brief add a tuple of a department and its capacity
    # @exception throws KeyError if department already in list
    # @param d is a department of type DeptT being added
    # @param n is the capacity of the department
    @staticmethod
    def add(d, n):
        for i in DCapALst.s:
            if i.dept == d:
                raise KeyError
        DCapALst.s = DCapALst.s + [DCapALst.__Tuple__(d, n)]

    @staticmethod
    ## @brief removes a tuple of a department and its capacity from list
    # @exception throws KeyError if trying remove a department not in list
    # @param d the department of type DeptT you want to remove
    def remove(d):
        raisee = True
        for i in DCapALst.s:
            if i.dept == d:
                raisee = False
                DCapALst.s.remove(i)
        if raisee:
            raise KeyError

    @staticmethod
    ## @brief tells you if a department is in the list
    # @param d department name you want to see if in the list
    # @return True if department is in list False if its not
    def elm(d):
        for i in DCapALst.s:
            if i.dept == d:
                return True
        return False

    @staticmethod
    ## @brief tells you the capacity of a department
    # @exception throws KeyError if trying to find capacity of department not in list
    # @param d the department of type DeptT you want to find the capacity of
    def capacity(d):
        for i in DCapALst.s:
            if i.dept == d:
                return i.cap
        raise KeyError
```

N Code for Partner's SALst.py

```
## @file SALst.py
# @author Seva Skvortsov
# @brief SALst
# @date 08/02/2019
from StdntAllocTypes import *
from AALst import *
from DCapALst import *


## @brief local tuple class to represent a student
class __StudentT__(NamedTuple):
    macid: str
    info: SInfoT


## @brief An abstract object to represent the Student Association List
class SALst:
    @staticmethod
    ## @brief local function to get gpa of a student
    # @param m string representing macid of student
    # @param s list of students
    # @return gpa of the student
    def ___get_gpa__(m, s):
        for i in s:
            if i.macid == m:
                return (i.info).gpa

    @staticmethod
    ## @brief initialize the list
    def init():
        SALst.s = []

    @staticmethod
    ## @brief adds a student to the list
    # @exception throws KeyError if student already in list
    # @param m string representing macid
    # @param i student information of type SInfoT
    def add(m, i):
        for t in SALst.s:
            if t.macid == m:
                raise KeyError
        newstudent = __StudentT__(m, i)
        SALst.s = SALst.s + [newstudent]

    @staticmethod
    ## @brief removes a student from the list
    # @exception throws KeyError if student not in list
    # @param m string representing macid of student to remove
    def remove(m):
        raise_e = True
        for i in SALst.s:
            if i.macid == m:
                raise_e = False
                SALst.s.remove(i)
        if raise_e:
            raise KeyError

    @staticmethod
    ## @brief tells you if a student is in the list
    # @param m string representing macid of student
    # @return True if student in list False if he is not
    def elm(m):
        for i in SALst.s:
            if i.macid == m:
                return True
        return False

    @staticmethod
    ## @brief tells you info of a student
    # @exception throws KeyError if student not in list
    # @param m string representing macid of student
    # @return info of the student of type SInfoT
    def info(m):
        for i in SALst.s:
            if i.macid == m:
                return i.info
```

```
        raise KeyError

    @staticmethod
    ## @brief sorts student by gpa and a function
    # @param f is the condition all student info (type SInfoT) has to pass
    # @return a list of macids sorted from highest to lowest gpa which passed condition
    def sort(f):
        outputlist = []
        for i in SALst.s:
            if f(i.info):
                outputlist = outputlist + [i]
        outputlist.sort(key=lambda i: i.info.gpa, reverse=True)

        outputlist = [i.macid for i in outputlist]
        return outputlist

    @staticmethod
    ## @brief calculates the average of a subset of the list of students
    # @exception Throws ValueError if no student passed the condition
    # @param f is the condition all student info (type SInfoT) has to pass
    # @return the GPA average of the subset of students
    def average(f):
        studentlist = []
        for i in SALst.s:
            if f(i.info):
                studentlist = studentlist + [i]
        if studentlist == []:
            raise ValueError
        else:
            gpa_sum = 0
            number_of_students = 0
            for i in studentlist:
                gpa_sum += i.info.gpa
                number_of_students += 1
            return gpa_sum / number_of_students

    ## @brief allocate all student from SALst object to AALst object
    # @exception throws RuntimeError if cannot allocate a student
    def allocate():
        AALst.init()
        f = SALst.sort(lambda t: t.freechoice and t.gpa >= 4.0)
        for m in f:
            ch = SALst.info(m).choices
            AALst.add_stdnt(ch.next(), m)
        s = SALst.sort(lambda t: (not (t.freechoice)) and t.gpa >= 4.0)
        for m in s:
            ch = SALst.info(m).choices
            alloc = False
            while ((not alloc) and (not ch.end())):
                d = ch.next()
                if AALst.num_alloc(d) < DCapALst.capacity(d):
                    AALst.add_stdnt(d, m)
                    alloc = True
            if not alloc:
                raise RuntimeError
```