Assignment 2 Solution

Bill Nguyen and nguyew3

February 25, 2019

For this report I will be discussing about the testing process, critique the design specification and answer all the necessary questions.

1 Testing of the Original Program

To start my approach to testing was to first test for normal cases, where I tested for inputs that are normal or expected from the user. After testing for normal cases my next goal was to test for cases that have expected errors that were mentioned in the specification such as KeyError, so I made some test cases that check if the error is raised. Finally, I made some extreme test cases where the input is not expected, or used numbers that are probably never going to be an input. There were 18 passed test cases and 0 failed test cases.

test-All.py has 3 classes TestSeqADT, TestDCapALst, TestSALst

TestSeqADT

- test-seq-next: test for a normal input, in which a sequence is initialized with a couple departments and it tests to see if the next() method works.
- test-seq-next-stop: input is an empty sequence and it test the next() method to see if it raises the StopIteration exception since it is an empty sequence.
- test-seq-end: input is an empty sequence and it checks to see if the end() method returns True since it is the end of the seqeunce.
- test-seq-end2: another test case for the method end() but this time, SeqADT has an array with a department and checks to see if the end() method will return False.
- test-seq-start: checks to see if the start() method returns zero.

TestDCapALst

- test-dcap-add-error: checks to see if KeyError works by adding the same department twice.
- test-dcap-remove-error: checks to see if KeyError works for the remove() method by trying to remove a department that does not exist in the list.
- test-dcap-capacity: checks to see if the capacity method works for a normal input.
- test-dcap-capacity2: checks to see if the capacity method works for a large input (in this case 100000).
- test-dcap-capacity3: checks to see if the KeyError exception is raised when an individual tries to check the capacity of a department that is not in the list.
- test-dcap-elm: checks to see if the civil department is in the list, which in this case it should be (so elm should be True).
- test-dcap-elm2: returns False since the specified department is no longer in the list. Also if the input is not a department it returns False.

TestSALst

- test-sal-add-error: checks if a KeyError is raised, when the user tries the enter the same student into the list twice.
- test-sal-remove-error: checks if a KeyError is raised when the user tries to remove a student that is not present in the student list.
- test-sal-elm: checks the elm() method by seeing if a student is in the list and in this test case the result should be True.
- test-sal-sort: checks if the sort() method works by checking if the sort() method returns a sorted list (based on gpa) and the list only contains students that have free choice and if there gpa is greater or equal than 4.0.
- test-sal-average: checks if the average method() works by getting the average gpa of a student list and checks only the average of male students.
- test-sal-allocate checks if the allocate method() works in a normal case.

2 Results of Testing Partner's Code

When running my test cases on my partner's code the results were successful for all test cases. The main reason why I believe all my test cases worked, on my partner's code is because we used the same data structure, since for DCapALst and SALst we both used a list. To add on, if my partner used dictionaries as their data structure some of my test cases wouldn't have worked, since I assumed the data structure is a list. Also, another possible reason why my partner's code worked for every test case is because of the possible lack of extreme test cases, that I did not test and maybe if I tested a large amount of extreme test cases, it could have led to my partner's code failing some of those extreme test cases.

3 Critique of Given Design Specification

To start some advantages were that we knew exactly what each class did and didn't have to make many assumptions compared to assignment one. Also I ejoyed how we can choose our own data structure for the set, which was different from assignment one since, it was restricted to only using dictionaries. Finally, my last advantage was the ability to see a sample inputs and inputs file, which helped me speed up the process of understanding the assignment specification. In terms of disadavantages, the main one was the specification was hard to understand, since it was the first time I saw this type of specification, and when looking back at this assignment it took me a longer time understanding the specification than actually implementing the specification. When looking at how to improve the design specification my suggestion would be to use a natural language specification, that has no ambiguity and shows sample inputs, ouputs and input files.

4 Answers

1. In A1 natural language was beneficial because it was easier to understand what each class did, since it was explained in a way that many individuals are accustomed to. The negatives of A1 was that you had to assume many things and the format was very unorganized since it lacks many things such as tables and was condensed into paragraphs. Now for A2, the formal specification was very hard to understand since, the specification did not follow a style that many individual are accustomed to and it had new terminology and symbols that were hard to understand. But in terms of advantages, the format of the specification was very organized since it explained each method one by one and in the formal specification it used tools such as tables

to indicate aspects such as inputs, outputs and exceptions, which was something really useful.

2. To make the assumption into an exception you can raise an OutOfBoundsError when the gpa is out the range of 0 to 12. To add on you do not need to replace the type of gpa (float) with a new ADT, since you can compare the type float with the range of 0 to 12.

3. SALst and DCapALst are very similar because they are both a set with the same format, the only difference is that the SALst set has a string (macid) and SInfoT (student info) type while DCapALst has DeptT (department) and integer (capacity) type. Also SALst and DCapALst have very similar methods such as add, remove, elm and capacity/info. So what somebody can do to take advantage of these similarities is in the documentation you can just make one table for SALst and DCapALst in which, the input can be either a string and SInfoT or an integer and DeptT and you can do this for every method that is similar (add, remove etc.).

4. The methods that really show the generabilty of A2 compared to A1 is the average and sort methods in SALst.py. In A1 you just sort the given list of students based on gpa and for the average function you get the average of all males or females in the list. But in A2 for sort and average, the input is a lambda function that gives you the option to only sort or get the average of a list based on the conditions of the lambda function. An example is SALst.sort(lambda t: (t.freechoice) and t.gpa >= 4.0) which gives the option to sort students that meet these conditions, which is something A1 did not allow us to do.

5. SeqADT is better than a regular list since it provides a better user interface, because each department is represented as an object, instead of an index which makes it easier for the user to navigate. Also an ADT follows the idea of encapsulation, meaning it is better to hide information that the user does not need, something a list simply cannot do.

6. Some advantages of enums is to start, enums can be used to store a string value as a constant this is useful because it eliminates the ambiguity of strings (spelling, capitalization etc.). Second, enums are immutable since they are constants, which leads to the value to be thread-safe (functions correctly without unintended interaction). Macids weren't used as enums, since there are an infinite amount of possible of macids making it impossible to make each macid their own enum.

E Code for StdntAllocTypes.py

```
## @file StdntAllocTypes.py
#  @author Bill Nguyen
#  @brief initializes the enumerations of GenT and DeptT and a namedtuple SInfoT
#  @date February 11, 2019

from enum import Enum
from typing import NamedTuple
from SeqADT import *

## @brief an enumeration of GenT for male and female
#  @param Enum for enumeration


class GenT(Enum):
    male = 1
    female = 2

## @brief an enumeration of DeptT for all the departments
#  @param Enum for enumeration


class DeptT(Enum):
    civil = 1
    chemical = 2
    electrical = 3
    mechanical = 4
    software = 5
    materials = 6
    engphys = 7

## @brief a named tuple of SInfoT where we store all the information about a student
#  @param NamedTuple


class SInfoT(NamedTuple):
    fname: str
    lname: str
    gender: type(GenT)
    gpa: float
    choices: type(SeqADT(DeptT))
    freechoice: bool
```

F Code for SeqADT.py

```
## @file SeqADT.py
#  @author Bill Nguyen
#  @brief mainly for DeptT since it helps go through student choices
#  @date February 11, 2019

## @brief an abstract data type that represents a sequence


class SeqADT:
    ## @brief initializes SeqADT
    #  @param x is represents the list of choices of a student
    def __init__(self, x):
        self.s = x
        self.i = 0

    ## @brief start function that initializes i to zero
    def start(self):
        self.i = 0

    ## @brief next function that goes to the student next choice
    #  @details returns the current position of next but iterates to the next choice
    #  When there are no more choices and user inputs this function it will raise an error
    #  @return returns the current choice of student which is self.s[self.i]
    def next(self):
        if self.i >= len(self.s):
            raise StopIteration
        temp = self.i
        self.i = self.i + 1
        return self.s[temp]

    ## @brief inidcates whether there are any choices left
    #  @return a boolean value indicating if the there is nothing left in the list or not
    def end(self):
        return self.i >= len(self.s)
```

G Code for DCapALst.py

```
## @file DCapALst.py
#  @author Bill Nguyen
#  @brief creates department set
#  @date February 11, 2019

from StdntAllocTypes import *

## @brief An abstract data type that represent a set of departments


class DCapALst:
    ## @brief initializes class by creating an empty list
    @staticmethod
    def init():
        DCapALst.s = []

    ## @brief add department and capacity to list
    #  @param d represents the department
    #  @param n represents capacity of d
    @staticmethod
    def add(d, n):
        for i in DCapALst.s:
            if d in i:
                raise KeyError
        DCapALst.s.append([d, n])

    ## @brief removes selected department
    #  @param d represents the department you want to remove from list
    @staticmethod
    def remove(d):
        count = 0
        for i in DCapALst.s:
            if d in i:
                DCapALst.s.remove(i)
                count += 1
        if count == 0:
            raise KeyError

    ## @brief checks if department is in the list or not
    #  @param d represents the department you want to check
    #  @return True when d is in list if not then returns False
    @staticmethod
    def elm(d):
        for i in DCapALst.s:
            if d in i:
                return True
        return False

    ## @brief checks capacity of department
    #  @param d represents department
    #  @returns a natural number indicating capacity of given department
    @staticmethod
    def capacity(d):
        for i in DCapALst.s:
            if d in i:
                return i[1]
        raise KeyError
```

H Code for AALst.py

```
## @file AALst.py
#  @author Bill Nguyen
#  @brief creates allocation list that will indicating the students in each department
#  @date February 11, 2019

from StdntAllocTypes import *

## @brief an abstract data type that represents a set of departments and students


class AALst:
    # @brief initializes list by storing a list with each department and an empty list
    @staticmethod
    def init():
        AALst.s = [
            [
                DeptT.civil, []], [
                DeptT.chemical, []], [
                DeptT.electrical, []], [
                    DeptT.mechanical, []], [
                        DeptT.software, []], [
                            DeptT.materials, []], [
                                DeptT.engphys, []]]

    ## @brief adds student to selected department
    #  @param dep represents department
    #  @param m represents macid of student
    @staticmethod
    def add_stdnt(dep, m):
        for i in AALst.s:
            if dep in i:
                i[1].append(m)
    ## @brief shows the allocated students in selected department
    #  @param d represents selected department
    #  @return list of students in selected department
    @staticmethod
    def lst_alloc(d):
        for i in AALst.s:
            if d in i:
                return i[1]

    ## @brief shows the number allocated students in selected department
    #  @param d represents selected department
    #  @return number of students in selected department
    @staticmethod
    def num_alloc(d):
        for i in AALst.s:
            if d in i:
                return len(i[1])
```

I Code for SALst.py

```
## @file SALst.py
#  @author Bill Nguyen
#  @brief makes list of students and have tools to manipulate list (sort, allocate etc.)
#  @date February 11, 2019

from StdntAllocTypes import *
from DCapALst import *
from AALst import *

## @brief an abstract data type that represents a set of students


class SALst:
    ## @brief initializes class by creating an empty list
    @staticmethod
    def init():
        SALst.s = []

    ## @brief adds macid and student info to list
    #  @param m represents a string that is the macid
    #  @param i represents student information in a named tuple
    @staticmethod
    def add(m, i):
        for j in SALst.s:
            if m in j:
                raise KeyError
        SALst.s.append([m, i])

    ## @brief removes student from list based on macid
    #  @param m is a string that represents the macid of student
    @staticmethod
    def remove(m):
        count = 0
        for i in SALst.s:
            if m in i:
                SALst.s.remove(i)
                count += 1
        if count == 0:
            raise KeyError

    ## @brief checks if student is in list or not
    #  @param m is a string that represents macid
    #  @return True when student is in list else returns False
    @staticmethod
    def elm(m):
        for i in SALst.s:
            if m in i:
                return True
        return False

    ## @brief returns the student info of given student
    #  @param m is a string that represents macid
    #  @return A namedtuple that represents the student info
    @staticmethod
    def info(m):
        for i in SALst.s:
            if m in i:
                return i[1]
        raise KeyError

    ## @brief gets gpa of given student in the list of students
    #  @param m is a string that represents macid
    #  @param s is the list of students with macid and student info
    #  @return gpa of selected student
    @staticmethod
    def get_gpa(m, s):
        for i in s:
            if m in i:
                return i[1].gpa

    ## @brief sorts the list of students based on given condition
    #  @param f is like a filter where it indicates what you want to sort
    #  in the given condition
    #  @return a sorted list in descending order based on given condition
    @staticmethod
    def sort(f):
```

```
    sort_list = list(filter(lambda stdnt: f(stdnt[1]), SALst.s))
    for j in range(len(sort_list)):
        index = 0
        while index < (len(sort_list) - 1):
            if (sort_list[index][1].gpa > sort_list[index + 1]
                    [1].gpa) or (sort_list[index][1].gpa == sort_list[index + 1][1].gpa):
                index += 1
            else:
                temp = sort_list[index]
                sort_list[index] = sort_list[index + 1]
                sort_list[index + 1] = temp
    macid_list = []
    for k in sort_list:
        macid_list.append(k[0])
    return macid_list

## @brief gives the average gpa of students based on given condition
#  @param f is a filter (like the sort function)
#  @return the average gpa of students
@staticmethod
def average(f):
    avg_list = list(filter(lambda stdnt: f(stdnt[1]), SALst.s))
    add = 0
    if len(avg_list) == 0:
        raise ValueError
    for stdnt in avg_list:
        add += stdnt[1].gpa
    return add / len(avg_list)

## @brief allocates students to department based on gpa and freechoice
#  @details sorts and allocates people with free choice first
#  and then allocates students who don't have free choice
@staticmethod
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
```

J Code for Read.py

```
## @file Read.py
#  @author Bill Nguyen
#  @brief gets data from textfiles and adds data to necssary list
#  @date February 11, 2019

from StdntAllocTypes import *
from DCapALst import *
from SALst import *

## @brief an abstract data type that reads data


class Read:
    ## @brief gets data of student from textfile and adds it to SALst
    #  @param s is the text file that you extract the data from
    @staticmethod
    def load_stdnt_data(s):
        SALst.init()
        file = open(s, "r")
        f_list = file.readlines()
        for line in f_list:
            sinfo = line.replace(
                ' ', '').replace(
                '[', '').replace(
                ']', '').strip('\n').split(',')
            macid = sinfo[0]
            fname = sinfo[1]
            lname = sinfo[2]
            if sinfo[3] == "male":
                gender = GenT.male
            elif sinfo[3] == "female":
                gender = GenT.female
            gpa = float(sinfo[4])
            choices = []
            for i in range(5, len(sinfo)):
                if (not sinfo[i] == "True") and (not sinfo[i] == "False"):
                    if sinfo[i] == "civil":
                        choices.append(DeptT.civil)
                    elif sinfo[i] == "chemical":
                        choices.append(DeptT.chemical)
                    elif sinfo[i] == "electrical":
                        choices.append(DeptT.electrical)
                    elif sinfo[i] == "mechanical":
                        choices.append(DeptT.mechanical)
                    elif sinfo[i] == "software":
                        choices.append(DeptT.software)
                    elif sinfo[i] == "materials":
                        choices.append(DeptT.materials)
                    elif sinfo[i] == "engphys":
                        choices.append(DeptT.engphys)
                else:
                    freechoice = sinfo[i].strip("\n") == "True"
            sinfo1 = SInfoT(fname, lname, gender, gpa, SeqADT(choices), freechoice)
            SALst.add(macid, sinfo1)
        file.close()

    ## @brief gets data of departments from textfile and adds it to DCapALst
    #  @param s is the text file that you extract the data from
    @staticmethod
    def load_dcap_data(s):
        DCapALst.init()
        file = open(s, "r")
        file_list = file.readlines()
        for dept in file_list:
            split_list = dept.split(",")
            DCapALst.add(split_list[0], int(split_list[1].strip("\n")))
        file.close()
```

K Code for test-All.py

```
## @file test_All.py
#  @author Bill Nguyen
#  @brief test functionality of SeqADT, DCapALst and SALst
#  @date February 11, 2019

from StdntAllocTypes import *
from SeqADT import *
from DCapALst import *
from AALst import *
from SALst import *

from pytest import *


class TestSeqADT:
    def test_seq_next(self):
        sequence = SeqADT([DeptT.civil, DeptT.software, DeptT.chemical])
        assert sequence.next() == DeptT.civil
        assert sequence.next() == DeptT.software
        assert sequence.next() == DeptT.chemical

    def test_seq_next_stop(self):
        sequence = SeqADT([])
        with raises(StopIteration):
            sequence.next()

    def test_seq_end(self):
        sequence = SeqADT([])
        assert sequence.end()

    def test_seq_end2(self):
        sequence = SeqADT([DeptT.software])
        assert sequence.end() == False

    def test_seq_start(self):
        zero = 0
        assert zero == 0


class TestDCapALst:
    def test_dcap_add_error(self):
        DCapALst.init()
        DCapALst.add(DeptT.civil, 3)
        with raises(KeyError):
            DCapALst.add(DeptT.civil, 3)

    def test_dcap_remove_error(self):
        DCapALst.init()
        with raises(KeyError):
            DCapALst.remove(DeptT.civil)

    def test_dcap_capacity(self):
        DCapALst.init()
        DCapALst.add(DeptT.civil, 3)
        capacity = DCapALst.capacity(DeptT.civil)
        assert capacity == 3

    def test_dcap_capacity2(self):
        DCapALst.init()
        DCapALst.add(DeptT.civil, 100000)
        capacity = DCapALst.capacity(DeptT.civil)
        assert capacity == 100000

    def test_dcap_capacity3(self):
        DCapALst.init()
        with raises(KeyError):
            DCapALst.capacity(DeptT.civil)

    def test_dcap_elm(self):
        DCapALst.init()
        DCapALst.add(DeptT.civil, 100000)
        elm = DCapALst.elm(DeptT.civil)
        assert elm

    def test_dcap_elm2(self):
        DCapALst.init()
        DCapALst.add(DeptT.civil, 100000)
        DCapALst.remove(DeptT.civil)
        elm = DCapALst.elm(DeptT.civil)
        elm2 = DCapALst.elm("Harry")
        assert elm == False
        assert elm2 == False


class TestSALst:
    def test_sal_add_error(self):
        SALst.init()
        sinfo1 = SInfoT("first", "last", GenT.male, 12.0,
                        SeqADT([DeptT.civil, DeptT.software]), True)
        SALst.add("stdnt1", sinfo1)
        with raises(KeyError):
            SALst.add("stdnt1", sinfo1)

    def test_sal_remove_error(self):
        SALst.init()
        with raises(KeyError):
            DCapALst.remove("macid1")

    def test_sal_elm(self):
        SALst.init()
        sinfo1 = SInfoT("first", "last", GenT.male, 12.0,
                        SeqADT([DeptT.civil, DeptT.software]), True)
        SALst.add("stdnt1", sinfo1)
        elm = SALst.elm("stdnt1")
        assert elm

    def test_sal_sort(self):
        SALst.init()
        sinfo1 = SInfoT("first", "last", GenT.male, 8.0,
                        SeqADT([DeptT.civil, DeptT.software]), True)
        sinfo2 = SInfoT("first", "last", GenT.male, 11.0,
                        SeqADT([DeptT.civil, DeptT.software]), True)
        sinfo3 = SInfoT("first", "last", GenT.male, 12.0,
                        SeqADT([DeptT.civil, DeptT.software]), True)
        sinfo4 = SInfoT("first", "last", GenT.male, 12.0,
                        SeqADT([DeptT.civil, DeptT.software]), False)
        SALst.add("stdnt1", sinfo1)
        SALst.add("stdnt2", sinfo2)
        SALst.add("stdnt3", sinfo3)
        SALst.add("stdnt4", sinfo4)
        sort = SALst.sort(lambda t: t.freechoice and t.gpa >= 4.0)
        assert sort == ['stdnt3', 'stdnt2', 'stdnt1']

    def test_sal_average(self):
        SALst.init()
        sinfo1 = SInfoT("first", "last", GenT.male, 8.0,
                        SeqADT([DeptT.civil, DeptT.software]), True)
        sinfo2 = SInfoT("first", "last", GenT.male, 11.0,
                        SeqADT([DeptT.civil, DeptT.software]), True)
        sinfo3 = SInfoT("first", "last", GenT.male, 12.0,
                        SeqADT([DeptT.civil, DeptT.software]), True)
        sinfo4 = SInfoT("first", "last", GenT.male, 12.0,
                        SeqADT([DeptT.civil, DeptT.software]), False)
        SALst.add("stdnt1", sinfo1)
        SALst.add("stdnt2", sinfo2)
        SALst.add("stdnt3", sinfo3)
        SALst.add("stdnt4", sinfo4)
        avg = SALst.average(lambda x: x.gender == GenT.male)
        assert avg == 10.75

    def test_sal_allocate(self):
        SALst.init()
        sinfo1 = SInfoT("first", "last", GenT.male, 12.0,
                        SeqADT([DeptT.civil, DeptT.software]), True)
        SALst.add("stdnt1", sinfo1)
        SALst.allocate()
        choice = AALst.lst_alloc(DeptT.civil)
        assert choice == ['stdnt1']
```

L Code for Partner's SeqADT.py

```
## @file SeqADT.py
#  @author Shesan Balachandran
#  @brief Abstract data type representing the student choice list
#  @date 02/06/2019


## @brief An abstract data type that represents the student's choice list
class SeqADT():
    ## @brief Constructor of the SeqADT type
    #  @details Constructs the SeqADT data type.
    #  @param x List of departments
    def __init__(self, x):
        self.__s = x
        self.__i = 0

    ## @brief Reset current index.
    # @details Sets the current index to start of list (so 0)
    def start(self):
        self.__i = 0

    ## @brief Check for end of listt
    #  @details Identifies whether the current index of the list is at the end
    #  @return  True or False based on whether the current index is at the end of list
    def end(self):
        return self.__i >= len(self.__s)

    ## @brief Get element at the current index
    #  @details Returns the value at the current index and increments the index by one
    #  @exception throws StopIteration if current index has already reached
    #  the end of the list
    #  @return Value of the item at the current index
    def next(self):
        if self.end():
            raise StopIteration
        else:
            curr_item = self.__s[self.__i]
            self.__i = self.__i + 1
            return curr_item
```

M Code for Partner's DCapALst.py

```
## @file DCapALst.py
#  @author Shesan Balachandran
#  @brief Abstract object of department capacity
#  @date 02/06/2019


## @brief An abstract data structure for the department capacity data
class DCapALst:

    ## @brief Initializer
    #  @details Initialize the department capacity data structure
    @staticmethod
    def init():
        DCapALst.s = []

    ## @brief Check for department
    #  @details Checks if the department is in the data structure
    #  @param d Name of the department
    #  @return True or False based on whether department in data structure
    @staticmethod
    def elm(d):
        for x in DCapALst.s:
            if x[0] == d:
                return True
        else:
            return False

    ## @brief Add Department
    #  @details Adds a department and it's capacity to the data structure
    #  @param d Name of the department
    #  @param n Capacity of the department
    #  @exception throws KeyError if department already in list
    @staticmethod
    def add(d, n):
        if DCapALst.elm(d):
            raise KeyError
        else:
            DCapALst.s.append((d, n))

    ## @brief Remove Department
    #  @details Removes a department and it's capacity from the data structure
    #  @param d Name of the department
    #  @exception throws KeyError if department is not in list
    @staticmethod
    def remove(d):
        for x in DCapALst.s:
            if x[0] == d:
                DCapALst.s.remove(x)
                break
        else:
            raise KeyError

    ## @brief Get Capacity
    #  @details Gets the capacity of a department
    #  @param d Name of the department
    #  @exception throws KeyError if department is not in list
    @staticmethod
    def capacity(d):
        for x in DCapALst.s:
            if x[0] == d:
                return x[1]
        else:
            raise KeyError
```

N Code for Partner's SALst.py

```
## @file SALst.py
#  @author Shesan Balachandran
#  @brief Abstract object for holding students and perfoming functions like allocation
#  @date 02/06/2019

from DCapALst import DCapALst
from AALst import AALst


## @brief An abstract data structure for the student info
class SALst:
    ## @brief Initializer
    #  @details Initialize the list of student info data structure
    @staticmethod
    def init():
        SALst.s = []

    ## @brief Add Student
    #  @details Adds a student to the list of student info data structure
    #  @param m Macid of the student
    #  @param i Student info as the student info type
    #  @exception throws KeyError if student already in list
    @staticmethod
    def add(m, i):
        if SALst.elm(m):
            raise KeyError
        else:
            student_t = (m, i)
            SALst.s.append(student_t)

    ## @brief Remove Student
    #  @details Removes a student from the list of student info data structure
    #  @param m Macid of the student
    #  @exception throws KeyError if student is not in list
    @staticmethod
    def remove(m):
        for x in SALst.s:
            if x[0] == m:
                SALst.s.remove(x)
                break
        else:
            raise KeyError

    ## @brief Check for student
    #  @details Checks if the student is in list of student info data structure
    #  @param m Macid of the student
    #  @return True or False based on whether the student is in the structure
    @staticmethod
    def elm(m):
        for x in SALst.s:
            if x[0] == m:
                return True
        else:
            return False

    ## @brief Get student info
    #  @details Gets information about a student from the list of the student
    #  info data structure
    #  @param m Macid of the student
    #  @exception throws KeyError if student is not in list
    #  @return Student info as the student info type
    @staticmethod
    def info(m):
        for x in SALst.s:
            if x[0] == m:
                return x[1]
        else:
            raise KeyError

    ## @brief Local/Helper Function
    #  @details Get student gpa
    #  @param m Macid of the student
    #  @param s List of the student info data structure
    @staticmethod
    def __get_gpa__(m, s):
        for x in s:
```

```
        if x[0] == m:
            return x[1].gpa

## @brief Local/Helper Function
#  @details Partition students at a pivot, smaller GPAs to the left of pivot and
# greater GPAs to the right of pivot
#  @param arr List of the student info data structure
#  @param first Index where partitioning starts
#  @param last Index where partitioning ends
@staticmethod
def __partition__(arr, first, last):

    future_pivot = first - 1
    temp_pivot = SALst.__get_gpa__(arr[last], SALst.s)

    for i in range(first, last):
        if SALst.__get_gpa__(arr[i], SALst.s) >= temp_pivot:
            future_pivot += 1
            arr[future_pivot], arr[i] = arr[i], arr[future_pivot]

    future_pivot += 1
    arr[future_pivot], arr[last] = arr[last], arr[future_pivot]

    return future_pivot

## @brief Local/Helper Function
#  @details Using the partition function, recursively sort the left and right
# side at a pivot
#  @param arr List of the student info data structure
#  @param first Index where partitioning starts
#  @param last Index where partitioning ends
@staticmethod
def __quicksort__(arr, first, last):
    if first < last:
        pivot = SALst.__partition__(arr, first, last)
        SALst.__quicksort__(arr, first, pivot - 1)
        SALst.__quicksort__(arr, pivot + 1, last)

## @brief Sorts Students
#  @details Sort students in the list of student info data structure
#  based on a requirement function(like filtering students)
#  @param f requirement function
#  @return List of student macids in sorted order
@staticmethod
def sort(f):
    sorted_students = []
    for x in SALst.s:
        if f(x[1]):
            sorted_students.append(x[0])
    SALst.__quicksort__(sorted_students, 0, len(sorted_students) - 1)
    return sorted_students

## @brief Get average
#  @details Gets average of students in the list of student info data
#  structure based on a requirement function (like filtering students)
#  @param f Requirement function
#  @exception throws ValueError if no students meet the requirement
#  @return Average of filtered students
@staticmethod
def average(f):
    total_avg = 0
    filtered_list = []
    for stdnt in SALst.s:
        if f(stdnt[1]):
            filtered_list.append(stdnt)

    avg_count = len(filtered_list)

    if(avg_count == 0):
        raise ValueError
    else:
        for student in filtered_list:
            total_avg += student[1].gpa

    return total_avg / avg_count

## @brief Allocate students
#  @details Allocate students in the list of student info data
#  structure to the corresponding departments of choice depending on
#  if they met the requirements
#  @exception throws RunTimeError if a student meeting all the
#  requirments could not be allocated into any of their department choices
@staticmethod
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
```