Assignment 2

Dominik Buszowiecki - buszowid

February 25, 2019

The purpose of this software design exercise is to write a Python program responsible for allocating first-year engineering students at McMaster University, into their second-year program using a formal specification. The program consists of the following files: AALst.py, DCapALst.py, Read.py, SALst.py, SeqADT.py, StdntAllocTypes.py, test_All.py, and Makefile.

1 Testing of the Original Program

The approach taken to test the required modules was to ensure that every line of code is correct and works as intended. To assist with this goal, the pytest coverage option was used to see the percentage of lines covered in every module. Due to the fact that most of the methods were quite small, it was easy to determine what test cases should be performed for each method.

There was a total of 30 test performed, the tests and their rational are written below.

SeqADT.py:

- SeqADT.__init__
    1. test_empty: Ensures that when we construct a object of type SeqADT using an empty list, the object also contains an empty list.
- SeqADT.start
    1. test_start: Ensures that when we run the start method, the index state variable gets reset to 0.
- SeqADT.next
    1. test_next: Tests that the next item in the sequence is correct.

2. test_stop_iteration: Checks if StopIteration is raised when there are no more items in the sequence.

- SeqADT.end

1. test_end_false: Tests if the end method returns False when there are items left in the sequence.
2. test_end_true: Tests if the end method return True if there are no items left in the sequence.

DCapALst.py:

- DCapALst.init

1. test_constructor: Tests if the constructor successfully creates an empty list.

- DCapALst.add

1. test_add: Ensures that when we add a department and its capacity, it appears in DCapAList.
2. test_add_exception: Tests if a KeyError is raised when we attempt to add a department that has been added already.

- DCapALst.remove

1. test_remove: Ensures that a department can be successfully removed from DCapALst.
2. test_remove_exception: Ensures that the remove method raises a KeyError if the department it is trying to remove is not in DCapALst.

- DCapALst.elm

1. test_elm_true: Tests that the elm method returns True if a department has already been added.
2. test_elm_false: Tests that the elm method returns False if a department has not been added.

- DCapALst.capacity

1. test_capacity: Ensures that the capacity method returns the correct capacity of a particular department.

2. test_capacity_exception: Tests if the capacity method raises a KeyError if the department given is not in DCapALst.

SALst.py:

- SALst.init
    1. test_constructor: Tests that the constructor successfully creates a empty list.
- SALst.add
    1. test_add: Tests that the add method successfully adds a students macid and their information into the student list (SALst).
    2. test_add_exception: Ensures that the add method raises a KeyError when we try to add a student that has already been added.
- SALst.remove
    1. test_remove: Ensures a student can be successfully deleted from SALst.
    2. test_remove_exception: Tests that the remove method raises a KeyError when we try to remove a student that is not in SALst.
- SALst.elm
    1. test_elm_true: Tests that the elm method returns True when a student is in SALst.
    2. test_elm_false: Tests that the elm method returns False when a student is not in SALst.
- SALst.info
    1. test_info: Ensures that the info method returns the correct information of a particular student.
    2. test_info_exception: Test that the info method raises a KeyError if we try to retrieve information of a student that is not in SALst.
- SALst.sort
    1. test_sort_empty: Tests that the sort method is able to return a empty list.
    2. test_sort: Tests that the sort method is able to read the filtering function correctly and sort only the students who pass the filter by GPA.

- SALst.average

    1. test_average: Tests that the average method can read the filtering function correctly and compute the average amongst the students who pass the filter.
    2. test_average_exception: Ensures that a ValueError is raised when we try to compute the average between 0 students (Due to no students passing the filter).

- SALst.allocate

    1. test_allocate_normal: A normal case for allocating students where some students: have free choice, get their first choice, and do not get their first choice. This test case ensures that students are allocated into the correct program.
    2. test_allocate_exception: Ensures that the allocate method will raise a RuntimeError if a student does not get into any of their choices because all their choices are full.

In the end, all test cases passed without any issues and pytest showed 100% coverage for all tested modules. However, I did discover a minor issue during the first round of testing. The issue occurred in the remove method of DCapALst.py, the issue was forgetting to add a return after I removed the student from the list. The lack of a return meant that the method would run the remaining lines of code after the deletion, therefore always raising a KeyError no matter what the input was. The problem however, was easy to debug because the modular design of the program allowed me to pinpoint it with ease.

2 Results of Testing Partner's Code

After running the test cases on my partner's code, all but one passed.

By looking through my partner's code, it was clear why there was such a high rate of success. This is because my partner's code was very similar (almost identical) to mine. The reason for the similarities is due to the use of a formal design specification which when understood correctly, can only be interpreted in one way.

The test case that failed was test_add in DCapALst. The reason for failure is due to the fact the my partner used dictionaries whereas I used a list of tuples to represent a department and its capacity. Therefore when test_add accessed the state variable directly, it expected a list and not a dictionary. With a minor modification to each method in DCapALst, I was able to get all the test cases to pass. The modification done was changing the state variable to a list and changing the indexing in each method to the one used with lists instead of dictionaries.

3 Critique of Given Design Specification

The thing I enjoyed about the given design specification is that it was not very ambiguous. The reason for this is due to the use of a formal design specification which uses mathematical notation to precisely define the program. The lack of ambiguity provided me with a greater degree of confidence that my implementation was correct.

What I disliked about the design is that I felt as though it was much less interesting to program in comparison to A1. The program was broken up into many small pieces that were defined formally making it obvious on the way it should be coded. This made me feel less creative as many design decisions were not imposed by me.

Another thing I did not like is reading the math notation in order to understand what a particular method should do. It seems backwards to me that we read the notation, convert it back to a natural language (in our mind) and once again write it formally into python. I also felt as though reading the math notation took much longer to understand than reading the natural language version in A1.

In order to improve the specification, my suggestion would be to somehow combine the natural design of A1 with the formal design of A2. Although it would likely take longer to write such a specification, it would be both less ambiguous and quicker to understand.

4 Answers

a) The main advantage with the use a formal design specification is the lack of ambiguity for the programmer. The lack of ambiguity results in less confusion and questions regarding the implementation. As mentioned before, when understood correctly, a formal design specification should only be interpreted in one way. The same cannot be said about a natural language specification This will give the one who created the specification a great deal of assurance that the program will work as he intended it to.

   One of the disadvantages that I see in a formal design specification is the background knowledge required to understand what is being asked of you. In order to read a MIS, you require some knowledge on discrete mathematics in order to determine what a particular method should do. Even if you have this knowledge, it could take a while to convert the given notation into something more understandable. If the specification used natural language, it would be easier to understand what the program should do.

   Another disadvantage to the given design specification is the lack of creativity allowed for the programmer. The specification is given in such a precisely defined way that the programmer often doesn't have to come up with their own algorithms. Instead, they just follow exactly what the specification says. In a natural language specification, it

is often up to the programmer to come up with the algorithms as well as make their own design decisions.

b) In my opinion, the best way to change the assumption that the GPA is between 0 and 12 to an exception is to modify the specification for the Read module. The change required would be checking if the GPA is in the correct range before adding a student to SALst in the load_stdnt_data method. If the student is not in the correct range, we simply raise an exception. The reason I believe this is the best way is because when the program is actually used, the user would first have to load student data from a file. By adding the exception in the Read module, we catch a mistake early and prevent it from carrying into the rest of the program.

   If the exception is incorporated in Read, the specification would not require you to replace a record type with a new ADT. This is because we raise the exception before we even add a student into the record type rather than inside the type.

c) If we ignore sort, average, and allocate in SALst and DCapALst, the modification required would be incorporating some form of inheritance between the classes. This could be done several ways, such as creating a general list class and having SALst and DCapALst inherit list. The list class would need to contain methods that both SALst and DCapALst share in common such as init, remove, and elm. Then, when we create DCapALst and SALst we only need to add on the methods that are specific to these classes such as capacity for DCapALst and allocate, average, sort and info for SALst.

d) A2 is more general than A1 in a few ways. One of the main ways this can be observed is in the average and sort function of the SALst module.

   In A1, the sort function returned a list of every student sorted by their GPA. However, in A2 the sort method is able to return a particular subset of students sorted by their GPA. This subset can be specified by the programmer.

   In A1, the average function was only able to compute the average amongst a particular gender whereas in A2, we are able to define what subset of students we want to compute the average amongst.

e) By using SeqADT instead of a regular list, we provide level encapsulation. Encapsulation means hiding the internal information of one piece of code from another. By using SeqADT, we are not able to access the list of choices directly, instead we must access it through the methods of the SeqADT class. The advantage to encapsulation is that you can be more confident that accidental changes cannot be made to the list as only acceptable changes defined by your methods are allowed.

Another advantage to using SeqADT rather than a list is it makes other functions easier to program and read. When we create a abstract data type like SeqADT, we often model it based on real world things. In this case we are modelling SeqADT after a sequence of values. This makes it much easier to program and read because the code will look closer to natural language.

f) The advantage of using Enums is to know exactly what the possible options are for a particular field. For example, if we wanted to know the possible engineering choices, we simply look at the documentation for StdntAllocTypes and observe the possible values. We also do not have to worry about spelling as the spelling is precisely defined. The fact that we don't have to worry about spelling benefits the programmer as they will likely encounter far less problems related to grammar.

The reason why Enums were not introduced in the specification for macids is because there are too many possible macid's. Remember, a Enum is able to map a particular string to a value, therefore we would need to have a enum for every macid. Every year there are new engineering students, therefore every year we would be required to change the enums for every macid. By knowing this, it would be much easier to simply use strings as Enums would make the software harder to maintain.

E Code for StdntAllocTypes.py

```
## @file StdntAllocTypes.py
#  @title Student Allocation Types
#  @author Dominik Buszowiecki
#  @date February 9, 2019

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


## @breif An Enumerated type of possible genders
class GenT(Enum):
    male = "male"
    female = "female"


## @brief An Enumerated type of possible engineering departments
class DeptT(Enum):
    civil = "civil"
    chemical = "chemical"
    electrical = "electrical"
    mechanical = "mechanical"
    software = "software"
    materials = "materials"
    engphys = "engphys"


## @brief A NamedTuple used to represent a student
#  @details A student has a: first name, last name, gender (given as a GenT type), gpa,
#           sequence of departments(given as a SeqADT of DeptT's), and a boolean to represent
#           if they have free choice.
class SInfoT(NamedTuple):
    fname: str
    lname: str
    gender: GenT
    gpa: float
    choices: SeqADT
    freechoice: bool
```

F Code for SeqADT.py

```
## @file SeqADT.py
#  @title Sequence ADT
#  @author Dominik Buszowiecki
#  @date February 9, 2019


## @brief An abstract data type that represents a sequence of values
class SeqADT:

    ## @brief SeqADT constructor
    #  @details Initializes the state variables of SeqADT. The state variables are a list that
    #           is given as a parameter and a variable used to index the list
    #           (initialized to 0).
    #  @param x A list of values
    def __init__(self, x: list):
        self.__s = x
        self.__i = 0

    ## @brief start will reset the index state variable to 0
    def start(self):
        self.__i = 0

    ## @brief next will return the next value in the sequence
    #  @exception throws StopIteration if there is no more items in the sequence
    #  @return value of next item in the sequence
    def next(self):
        if self.__i >= len(self.__s):
            raise StopIteration
        self.__i += 1
        return self.__s[self.__i - 1]

    ## @breif end will check if there are more items in the sequence
    #  @return True if there are no more items in the sequence, otherwise False
    def end(self) -> bool:
        return self.__i >= len(self.__s)
```

G Code for DCapALst.py

```
## @file DCapALst.py
#  @title Department Capacity Association List
#  @author Dominik Buszowiecki
#  @date February 9, 2019

from StdntAllocTypes import *


## @brief An abstract data type containing the capacities of engineering departments as a list
class DCapALst:

    ## @brief Initializes the Department Capacity List to be empty
    @staticmethod
    def init():
        DCapALst.s = []

    ## @brief Adds a department and its capacity to the list
    #  @exception throws KeyError if the given department has been added before
    #  @param d A department of type StdntAllocTypes.DeptT
    #  @param n An integer representing the capacity of the department (d parameter)
    @staticmethod
    def add(d: DeptT, n: int):
        for i in DCapALst.s:
            if d == i[0]:
                raise KeyError
        DCapALst.s.append((d, n))

    ## @breif Removes a department and its capacity from the list
    #  @exception throws KeyError if the given department is not in DCapALst
    #  @param d A department of type StdntAllocTypes.DeptT to be removed
    @staticmethod
    def remove(d: DeptT):
        for i in range(0, len(DCapALst.s)):
            if d == DCapALst.s[i][0]:
                del DCapALst.s[i]
                return
        raise KeyError

    ## @brief elm checks if a department has been added
    #  @param d A department of type StdntAllocTypes.DeptT
    #  @return True if the department has been added, otherwise False
    @staticmethod
    def elm(d: DeptT) -> bool:
        for i in DCapALst.s:
            if d == i[0]:
                return True
        return False

    ## @brief capacity returns the capacity of a department
    #  @exception throws KeyError if the department given is not in DCapALst
    #  @param d A department of type StdntAllocTypes.DeptT
    #  @return An integer representing the capacity of the department given as a parameter.
    @staticmethod
    def capacity(d: DeptT) -> bool:
        for i in DCapALst.s:
            if d == i[0]:
                return i[1]
        raise KeyError
```

H Code for AALst.py

```
## @file AALst.py
#  @title Allocation Association List Module
#  @author Dominik Buszowiecki
#  @date February 9, 2019

from StdntAllocTypes import *


## @brief An abstract data type containing engineering departments and the students allocated
#         into them
class AALst:

    ## @brief Initiazlies the AALst
    #  @details The list is initialized with each department and an empty list of students for
    #   each department.
    @staticmethod
    def init():
        AALst.s = []
        for i in DeptT:
            AALst.s.append((i, []))

    ## @details add_stdnt adds a student to a specific department
    #  @param dep A department of type StdntAllocTypes.DeptT
    #  @param m A string representing the students macid
    @staticmethod
    def add_stdnt(dep: DeptT, m: str):
        for i in AALst.s:
            if i[0] == dep:
                i[1].append(m)

    ## @details lst_alloc returns a list of students in a specific department
    #  @param d A department of type  StdntAllocTypes.DeptT
    #  @return A list of strings where each string is a macid
    @staticmethod
    def lst_alloc(d: DeptT) -> list:
        for i in AALst.s:
            if i[0] == d:
                return i[1]

    ## @details num_alloc returns the number of students in a department
    #  @param d A department of type  StdntAllocTypes.DeptT
    #  @return A integer representing the number of students in a department
    @staticmethod
    def num_alloc(d: DeptT) -> int:
        for i in AALst.s:
            if i[0] == d:
                return len(i[1])
```

I Code for SALst.py

```
## @file SALst.py
#  @title Student Association List
#  @author Dominik Buszowiecki
#  @date February 9, 2019

from StdntAllocTypes import *
from AALst import *
from DCapALst import *
from typing import Callable


## @brief An abstract data type of all first year engineerng students
class SALst:

    ## @brief init initializes the list of students to be empty
    @staticmethod
    def init():
        SALst.s = []

    ## @brief Adds a student into the SALst
    #  @exception throws KeyError if the student given has been added before
    #  @param m A string of a student's macid
    #  @param i Information of a student given with the data type StdntAllocTypes.SInfoT
    @staticmethod
    def add(m: str, i: SInfoT):
        for student in SALst.s:
            if student[0] == m:
                raise KeyError
        SALst.s.append((m, i))

    ## @brief Removes a student from the SALst
    #  @exception throws KeyError if a student to be removed is not found
    #  @param m A string of a student's macid
    @staticmethod
    def remove(m: str):
        for i in range(0, len(SALst.s)):
            if SALst.s[i][0] == m:
                del SALst.s[i]
                return
        raise KeyError

    ## @brief elm checks if a student is already in the SALst
    #  @param m A string of a student's macid
    #  @return True if a student is in SALst, otherwise False
    @staticmethod
    def elm(m: str):
        for student in SALst.s:
            if student[0] == m:
                return True
        return False

    ## @brief returns the information assoaciated with a student
    #  @exception throws KeyError if the student is not found
    #  @param m A string of a student's macid
    #  @return A students information with the type StdntAllocTypes.SInfoT
    @staticmethod
    def info(m: str) -> SInfoT:
        for student in SALst.s:
            if student[0] == m:
                return student[1]
        raise KeyError

    ## @brief Sorts a subset of students based on GPA
    #  @details The method is given a function that is able to filter a student. The filter
    #           function takes in a student (SInfoT) and returns True if they pass the filter.
    #           The method will return a list of macids that passed the filter, sorted by
    #           their GPA in descending order.
    #  @param f A filtering function that returns a boolean
    #  @return A list of strings (each string is a macid) sorted by their GPA in
    #          descending order
    @staticmethod
    def sort(f: Callable[[], bool]) -> list:
        temp_l = []
        for student in SALst.s:
            if f(student[1]):
                temp_l.append(student)
```

```
    temp_l = sorted(temp_l, key=lambda gpa_student: gpa_student[1].gpa, reverse=True)
    sorted_list = []
    for i in temp_l:
        sorted_list.append(i[0])
    return sorted_list

## @brief Computes the average of a particular subset of students
#  @details The method is given a function that is able to filter a student. The function
#           takes in a student(SInfoT) and returns True if they pass the filter. The
#           method will then compute the average GPA amongst students who passed the
#           filter.
#  @exception throws ValueError if there are no students that pass the filter function.
#  @param f A filtering function that returns a boolean
#  @return A float representing the average GPA amongst a subset of students
@staticmethod
def average(f: Callable[[], bool]) -> float:
    i = 0
    size = 0
    for student in SALst.s:
        if f(student[1]):
            i += student[1].gpa
            size += 1
    if size == 0:
        raise ValueError
    else:
        return i / size

## @breif Allocates students in SALst into their program
#  @details Students are allocated into a department in AALst.
#  Students with free choice are allocated first. The remaining students are allocated in
#  a order based on their GPA, a student is allocated into their highest preferred choice
#  that is not full in capacity.
#  @exception throws RuntimeError if all of a student's choices are full.
@staticmethod
def allocate():
    AALst.init()
    f = SALst.sort(lambda t: t.freechoice and t.gpa >= 4.0)
    for student in f:
        ch = SALst.info(student).choices
        AALst.add_stdnt(ch.next(), student)

    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 test$_{A}ll.py$

```
## @file test_All.py
#  @title Program Unit Testing
#  @author Dominik Buszowiecki
#  @date February 9, 2019

import pytest
from SeqADT import *
from DCapALst import *
from SALst import *
from StdntAllocTypes import *


class TestSeqADT:

    def setup_method(self):
        self.new = SeqADT(["test1", "test2", "test3"])

    def teardown_method(self):
        self.new = None

    def test_empty(self):
        new = SeqADT([])
        with pytest.raises(StopIteration):
            new.next()

    def test_stop_iteration(self):
        new = SeqADT(["Test1"])
        new.next()
        with pytest.raises(StopIteration):
            new.next()

    def test_next(self):
        assert self.new.next() == "test1"

    def test_end_false(self):
        assert not self.new.end()

    def test_end_true(self):
        self.new.next()
        self.new.next()
        self.new.next()
        assert self.new.end()

    def test_start(self):
        self.new.next()
        self.new.start()
        assert self.new.next() == "test1"


class TestDCapALst:

    def setup_method(self):
        DCapALst.init()

    def teardown_method(self):
        DCapALst.s = None

    def test_constructor(self):
        DCapALst.s == []

    def test_add(self):
        DCapALst.add(DeptT.engphys, 10)
        assert DCapALst.s == [(DeptT.engphys, 10)]

    def test_add_exception(self):
        DCapALst.add(DeptT.software, 20)
        with pytest.raises(KeyError):
            DCapALst.add(DeptT.software, 30)

    def test_elm_true(self):
        DCapALst.add(DeptT.software, 10)
        assert DCapALst.elm(DeptT.software)

    def test_elm_false(self):
        DCapALst.add(DeptT.chemical, 100)
        assert not DCapALst.elm(DeptT.software)
```

```
    def test_remove(self):
        DCapALst.add(DeptT.chemical, 20)
        DCapALst.remove(DeptT.chemical)
        assert DeptT.chemical not in DCapALst.s

    def test_remove_exception(self):
        DCapALst.add(DeptT.materials, 30)
        with pytest.raises(KeyError):
            DCapALst.remove(DeptT.electrical)

    def test_capacity(self):
        DCapALst.add(DeptT.civil, 100)
        assert DCapALst.capacity(DeptT.civil) == 100

    def test_capacity_exception(self):
        DCapALst.add(DeptT.mechanical, 100)
        with pytest.raises(KeyError):
            DCapALst.capacity(DeptT.electrical)


class TestSALst:

    def setup_method(self):
        SALst.init()
        SALst.add("macid1", SInfoT("first1", "last1", GenT.male, 12.0,
                                   SeqADT([DeptT.civil, DeptT.chemical]), True))
        SALst.add("macid2", SInfoT("first2", "last2", GenT.male, 11.0,
                                   SeqADT([DeptT.civil, DeptT.chemical]), False))
        SALst.add("macid3", SInfoT("first3", "last3", GenT.male, 10.0,
                                   SeqADT([DeptT.chemical, DeptT.civil]), True))
        SALst.add("macid4", SInfoT("first4", "last4", GenT.female, 11.5,
                                   SeqADT([DeptT.civil, DeptT.chemical]), True))

    def teardown_method(self):
        SALst.s = None

    def test_constructor(self):
        SALst.init()
        assert SALst.s == []

    def test_add(self):
        SALst.init()
        sinfo1 = SInfoT("first", "last", GenT.male, 12.0,
                        SeqADT([DeptT.civil, DeptT.chemical]), True)
        SALst.add("macid1", sinfo1)
        assert SALst.s == [("macid1", sinfo1)]

    def test_add_exception(self):
        sinfo1 = SInfoT("first", "last", GenT.male, 12.0,
                        SeqADT([DeptT.civil, DeptT.chemical]), True)
        with pytest.raises(KeyError):
            SALst.add("macid1", sinfo1)

    def test_remove(self):
        SALst.remove("macid3")
        assert "macid3" not in SALst.s

    def test_remove_exception(self):
        with pytest.raises(KeyError):
            SALst.remove("macid")

    def test_elm_true(self):
        assert SALst.elm("macid1")

    def test_elm_false(self):
        assert not SALst.elm("macid")

    def test_info(self):
        sinfo = SALst.info("macid1")
        sinfo1 = SInfoT("first1", "last1", GenT.male, 12.0,
                        SeqADT([DeptT.civil, DeptT.chemical]), True)
        assert sinfo[0] == sinfo1[0]
        assert sinfo[1] == sinfo1[1]
        assert sinfo[2] == sinfo1[2]
        assert sinfo[3] == sinfo1[3]
        assert sinfo[4].next() == sinfo1[4].next() and sinfo[4].next() == sinfo1[4].next()
        assert sinfo[5] == sinfo1[5]

    def test_info_exception(self):
```

```
    with pytest.raises(KeyError):
        SALst.info("macid")

def test_sort_empty(self):
    assert SALst.sort(lambda t: t.gpa > 12) == []

def test_sort(self):
    assert SALst.sort(lambda t: t.freechoice and t.gpa >= 4.0) == ["macid1",
                                                                   "macid4",
                                                                   "macid3"]

def test_average(self):
    assert SALst.average(lambda x: x.gender == GenT.male) == 11.0

def test_average_exception(self):
    with pytest.raises(ValueError):
        assert SALst.average(lambda x: x.gpa > 12)

def test_allocate_normal(self):
    DCapALst.init()
    DCapALst.add(DeptT.civil, 3)
    DCapALst.add(DeptT.chemical, 2)
    SALst.allocate()
    assert AALst.s == [(DeptT.civil, ["macid1", "macid4", "macid2"]),
                       (DeptT.chemical, ["macid3"]),
                       (DeptT.electrical, []),
                       (DeptT.mechanical, []),
                       (DeptT.software, []),
                       (DeptT.materials, []),
                       (DeptT.engphys, [])]

def test_allocate_exception(self):
    DCapALst.init()
    DCapALst.add(DeptT.civil, 2)
    DCapALst.add(DeptT.chemical, 0)
    with pytest.raises(RuntimeError):
        SALst.allocate()
```

K Code for Read.py

```
## @file Read.py
#  @title Read
#  @author Dominik Buszowiecki
#  @date February 9, 2019

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


## @brief Loads students from a file into the SALst
#  @details Each line in the file represents a student.
#           The format of each line should be: \n
#         macid, firstname, lastname, gender, gpa, [choice1, choice2, ...], freechoice \n
#           where gpa is a real number, gender is either male or female
#           and freechoice is either True or False.
#  @param s A string representing the name of the file
def load_stdnt_data(s: str):
    stdnt_data = open(s, 'r')
    student_list = stdnt_data.read().splitlines()
    stdnt_data.close()

    SALst.init()
    for student in student_list:
        info_list = re.split(", \[|\], |, ", student)  # noqa: W605
        choices = []
        for i in range(5, len(info_list) - 1):
            choices.append(DeptT(info_list[i]))
        if info_list[-1] == "False":
            free_choice = False
        else:
            free_choice = True
        sinfo = SInfoT(info_list[1], info_list[2],  # Students fname and lname
                       GenT(info_list[3]),          # Students Gender
                       float(info_list[4]),         # Students GPA
                       SeqADT(choices),             # Students choices
                       free_choice)                 # Freechoice boolean
        SALst.add(info_list[0], sinfo)


## @brief Loads department capacities from a file into the DCapALst
#  @details Each line in the file represents a department.
#           The format of each line should be: \n
#           department_name, capacity \n
#           where capacity is an integer.
#  @param s A string representing the name of the file
def load_dcap_data(s: str):
    dept_capacity = open(s, 'r')
    department_list = dept_capacity.read().splitlines()
    dept_capacity.close()

    DCapALst.init()
    for department in department_list:
        dep_list = department.split(', ')
        DCapALst.add(DeptT(dep_list[0]), int(dep_list[1]))
```

L Code for Partner's SeqADT.py

```
## @file SeqADT.py
#  @title SeqADT
#  @author Charles Zhang
#  @February 11, 2019


## @brief Allows for easier readability of a list of choices of departments.
#  @details This class allows for creations of SeqADT typed objects. It has functions
#           that will interate through the choices.
class SeqADT:

    ## @brief constructor
    #  @details sets s and i private attribute of self
    #  @param self self parameter
    #  @param x list of department choices
    def __init__(self, x):
        self.__s = x
        self.__i = 0

    ## @brief sets i attribute to 0
    #  @param self self parameter
    def start(self):
        self.__i = 0

    ## @brief iterates through the list of choices
    #  @details returns current element and iterates to the next element in the list
    #  @param self self parameter
    #  @return current element
    def next(self):
        if (self.__i >= len(self.__s)):
            raise StopIteration
        self.__i += 1
        return self.__s[self.__i - 1]

    ## @brief checks if i reached the end of the list
    #  @param self self parameter
    #  @return boolean value on whether or not its the end of the list
    def end(self):
        return self.__i >= len(self.__s)
```

M Code for Partner's DCapALst.py

```
## @file SeqADT.py
#  @title SeqADT
#  @author Charles Zhang
#  @February 11, 2019


## @brief abstract object for department capacity
#  @details object with ability to add new departments,
#           remove department, and get information about a department.
class DCapALst:

    s = {}

    ## @brief constructor function.
    #  @details Sets .s attribute to an empty dictionary
    @staticmethod
    def init():
        DCapALst.s = {}

    ## @brief adds a new department to the object
    #  @details adds a new key and keyvalue. The key is the department
    #           name and the key value is the capacity
    #  @param d department
    #  @param n capacity
    @staticmethod
    def add(d, n):
        if (d in DCapALst.s):
            raise KeyError
        DCapALst.s[d] = n

    ## @brief removes a department and its key value
    #  @param d department
    @staticmethod
    def remove(d):
        if (d not in DCapALst.s):
            raise KeyError
        else:
            del DCapALst.s[d]

    ## @brief checks if a department is in the dictionary or not
    #  @param d department
    #  @return boolean value indicating whether or not the department is in the dictionary
    @staticmethod
    def elm(d):
        return d in DCapALst.s

    ## @brief gives info on the capacity of a given department
    #  @param d department
    #  @return capacity of the given department
    @staticmethod
    def capacity(d):
        if (d not in DCapALst.s):
            raise KeyError
        else:
            return DCapALst.s.get(d)
```

N Code for Partner's SALst.py

```
## @file SeqADT.py
#  @title SeqADT
#  @author Charles Zhang
#  @February 11, 2019
from StdntAllocTypes import *
from AALst import *
from DCapALst import *


## @brief abstract object that allocates students into the appropriate department
#  @details implements several functions such as add and info for the allocation()
#   function to work and allocate students.
class SALst:

    s = []

    ## @brief initialize attribute s with an empty list
    @staticmethod
    def init():
        SALst.s = []

    ## @brief adds a student into the list s to be allocated
    #  @param m macid
    #  @param i student info
    @staticmethod
    def add(m, i):
        for k in SALst.s:
            if (k[0] == m):
                raise KeyError
        SALst.s.append((m, i))

    ## @brief removes a given student from the list s
    #  @param m macid
    @staticmethod
    def remove(m):
        not_in_list = True
        for k in SALst.s:
            if (k[0] == m):
                SALst.s.remove(k)
                not_in_list = False
        if (not_in_list):
            raise KeyError

    ## @brief tells whether or not a given macid is in the list s
    #  @param m macid
    #  @return boolean value on whether or not given macid is in the list
    @staticmethod
    def elm(m):
        in_list = False
        for k in SALst.s:
            if (k[0] == m):
                in_list = True
        return in_list

    ## @brief gives info of student based on given macid
    #  @param m macid
    #  @return info of a student based on the given macid
    @staticmethod
    def info(m):
        not_in_list = True
        for k in SALst.s:
            if (k[0] == m):
                not_in_list = False
                return (k[1])
        if (not_in_list):
            raise KeyError

    ## @brief sorts the list of students in descending order based on given function
    #  @details appends students that satisfies given function into a list L and then
    #   sorts the list L in descending order based on gpa
    #  @param f function used to filter students out
    #  @return list L that is sorted in descending order and satisfies f
    @staticmethod
    def sort(f):
        ls = []
        for k in SALst.s:
```

```
            if f(k[1]):
                ls.append(k[0])

        for i in range(len(ls)):
            for j in range(0, (len(ls)) - i - 1):
                if SALst.__get_gpa(ls[j], SALst.s) < SALst.__get_gpa(ls[j + 1], SALst.s):
                    ls[j], ls[j + 1] = ls[j + 1], ls[j]

        return (ls)

    ## @brief gives average of students that satisfies the f
    #  @param f function that is used to filter students that satisfies its conditions
    #  @return average
    @staticmethod
    def average(f):
        fset = []
        total_gpa = 0
        for k in SALst.s:
            if f(k[1]):
                fset.append(k[1])

        if (len(fset) == 0):
            raise ValueError

        for k in fset:
            total_gpa += k.gpa

        return total_gpa / len(fset)

    ## @brief allocates students into the right departments
    #  @details free choice students with a gpa over 4 gets allocated first.
    #           Rest of the students gets allocated in order based on their gpa.
    #           If the department gets filled the student gets assigned of its next choice.
    @staticmethod
    def allocate():
        AALst.init()
        free = SALst.sort(lambda t: t.freechoice and t.gpa >= 4.0)
        for m in free:
            ch = SALst.info(m).choices
            AALst.add_stdnt(ch.next(), m)

        not_free = SALst.sort(lambda t: (not t.freechoice) and t.gpa >= 4.0)
        for m in not_free:
            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

    ## @brief local function to get the gpa of a macid
    #  @param m macid
    #  @param s list of tuples the macid belongs in
    #  @return gpa of the student with the given macid
    @staticmethod
    def __get_gpa(m, s):
        for k in s:
            if (k[0] == m):
                return k[1].gpa
```