Assignment 2 Solution

Donisius Wigie

February 23, 2019

The modules created are to allocate first year engineering students into their second year programs based on their top choices and gpa. The problem is the same as in assignment 1 but the approach taken is much different. We were given formal specifications as opposed to the natural language specifications given in assignment 1.

1 Testing of the Original Program

The first module I began testing was the DCapALst module. After creating all my test cases for this module, my module one test for the add method. When I was first creating the method, I had misunderstood the exception. The exception is if the department already existed, then to raise an exception. The way I interpreted that in the beginning was to raise an exception when something other than an integer value was given as the second argument. I caught this while testing for the exception for the add method and changed my code immediately. My test cases for the elm function were to see if they returned false if the department did not exist and true if it did. Before I tested elm for a department which already existed, I needed to test the add function to make sure it worked properly so that I would be able to use the add function to test the elm function. I followed this principle as I made my test cases for other methods where I test the simplest methods first before using those simple methods to test more complicated methods. The second module I tested was my SeqADT module. I began my testing the constructor and the best way I felt to do that was to see if the type instantiated was of type SeqADT using the issinstance method. While creating my test case for the exception specified for the next() method, I realized that my next method did not raise an exception when the index passed the bounds, it stopped itself before it was able to raise an exception. I changed my code to match the specifications. I tested start by making sure when it is called the index goes back to 0 and I tested end to see if it would return true if it was at the end of the list and false if not. Finally, I tested SALst. I began with testing the remove function to make sure the exception gets raised when you try to remove something that isn't in the list. I

then tested the elm method using the normal cases (true if something exists false if not) and used the elm method to test the add function to make sure when you add something it exists in the list. I tested the info method to make sure the correct type was being returned (SInfoT) using the issinstance method and that the info stored in SInfoT was correct by comparing them to the expected information. I tested the sort method to make sure a sample array was sorted and would return the correct list of arrays according to 2 different lambda functions passed in. I tested the average function by calling the method and comparing the returned value to the expected result and I tested the exception by causing a division by zero to occur. I tested the allocate function by first seeing if the exception would be raised if there were no departments available. Then I provided all the expected information and tested allocate by comparing the students allocated into the departments with the expected students allocated to the departments. I then tested allocate to see if it would raise an exception when a student does not get allocated into any program.

2 Results of Testing Partner's Code

My partner's code passed every single one of my test cases. I first thought it was strange for all the test cases to pass because for the first assignment my partner passed only 7 out of the 18 tests. I expected more test cases to pass for this assignment since the specifications were very strict, but I did not expect all the test cases to pass. After closer inspection of my partner's code it is very clear why it passed every test case. I created by test case not by looking back at the code I wrote, but by looking at the specifications provided for each of the methods we were supposed to test from the assignment document. My partner's code very closely followed the specifications and was able to raise exceptions for the right conditions. They also implemented their SALst and DCapALst as a list when the module is instantiated. Our code looks very similar since we both seemed to follow the specifications very closely, and this showed with the passing of all my test cases. This is a contrast from the first assignment where my partner's code looked nothing like mine and we both made drastically different assumptions and therefore was hard to anticipate how to make test cases for another person's code. For this assignment, we were not able to make nearly as much assumptions and therefore was more likely that the test cases created would work for other people's code as well.

3 Critique of Given Design Specification

I grew an appreciation for the design as everything was so modularized. The methods in the classes worked together perfectly and since it was so modularized, when something

went wrong it was very easy to isolate the problem. Furthermore, I did not need to worry about creating bigger problems in other parts of the program since the problem was isolated. However, as much as I enjoy modularization and appreciate its benefits, I do believe that too much modularization can be redundant. I felt like some parts of the specifications were redundant such as the next() and end() methods in the SeqADT module. I think that it would be better to have end built into next to that it stops when it is at the end instead of raising an exception. It can be confusing when trying to use the two together to make sure it isn't at the end by calling the end method and then calling the next method. The user might find that redundant and confusing. I also did not understand the use of the get_gpa method in the SALst module. Why use a method to get the gpa when you are able to do it easily without a method with .gpa on a SInfoT type.

4 Answers

1. The natural language specification of A1 were very different than the formal specification of A2. With the natural language specification, there were many assumptions that needed to be made. I needed to make assumptions on when exceptions should be raised. I also needed to assumptions on how to deal with certain conditions and events. Some of these assumptions include dealing with the event that a student does not get allocated into any department, how to deal with/prevent a DivisionByZero exception, how to deal with nonsensical inputs, how to deal with the boundary cases, and many others. To summarize, with the natural language specification of A1, there were many assumptions I needed to make as the specification did not include how certain conditions or sequence of events should be dealt with. An advantage to using natural language specifications would be that it is much easier than formal specification to understand. The simplicity comes from using English to communicate the specifications so it is easier for the programmer to understand what the specification is asking for. Contrasting the natural language specification with the formal specification of A2, there were very little assumptions to be made as I created the modules for A2. I did not have to make assumptions about when exceptions should be raised because it specifies when they should be raised. The specification provides most assumptions I possibly would make such as in the SALst module. The assumptions given were that SALst.init() is called before any other access program and DCapALst has been populated before allocate() is ran. A disadvantage to using formal specification is that it can be difficult to understand what the specification is asking for. It is difficult to adjust to reading mathematical specifications and it is also more difficult to express what you want the program to

do through mathematical specifications rather than natural language.

2. Since the specification makes the assumption that the gpa will be between 0 and 12, I did not include any way to deal with gpa inputs which are not within that range. It is entirely possible for someone to use the program and input a gpa which is less than 0 or greater than 12 which can cause issues in the output of the program when allocate is called. The way I would modify this assumption is to create an exception when the program encounters a gpa which is less than 0 or greater than 12, most likely in the sort() method. When the sort method comes across a student who has a gpa that is below 0 or greater than 12, then the exception should be raised to indicate this. I don't necessarily need to modify the specification in order to replace a record type with a new ADT since the exception would be raised when sort comes across this but it may be beneficial to go this route. This is because a new ADT to replace one of the record types like SInfoT can be created to catch such exceptions before the data gets put through the rest of the program. This way, potential problems which may arise can be isolated and caught early on.

3. The two modules SALst and DCapALst are very similar in that they both contain the methods add, remove and elm, which have very similar functions. The two functions info(m) from SALst and capacity(d) from DCapALsy also have similar functions since they return information about a member of each respective list. The documentation can be modified to take advantage of these similarities by incorporating just one module containing student information and the department capacities so that the functions can be used to add, remove, see if an element exists and return information about an entry all in one module with one method for each function. This would of course reduce the modularization of the program, but it would also reduce the amount of confusion since two modules wouldn't have the same functions with the same name. Another way we can take advantage of the similarities is by creating a separate module containing the add, remove, elm and info/capacity methods so that it can be used in the SALst and DCapALst. Doing it this way will increase the level of modularization in the program.

4. In several ways, A2 is more general than A1 because it is able to solve more general problems. For example, in A1 the students each must three top picks and the problem is allocate the students based on those three top picks. In A2, the students are able to have as many picks as possible and the program will allocate the students according to their picks regardless of how many choices they have with the assumption that it isn't zero choices, since that would raise an exception when allocate is called. Students also do not need to have the same number of choices and the program will still be able to allocate the students. The problem when from allocate

the students based off their top three choices to allocate the students based off their top n choices.

5. Since we are representing the list of choices for each student by a custom object, SeqADT instead of a Python list, there are many advantages that come along with this design choice. Using SeqADT to represent the list of choices for each student, we can create built in methods to process the information stored. We created a method which can set the index back to the beginning, a method to return the element which is currently indexed and increment the index, and we have a method which checks to see if the list is at the end. Because of this ability to create built in methods to help process the information it is easier for methods outside of the module to process the information. Furthermore, with this design choice, it is much easier for changes to the number of choices featured in a students top choices to be implemented. This is because we only need to keep checking whether we are at the end of the list or not using the end method and because of this it is easier to create outside methods to accommodate for any changes.

6. Using enums to represent student gender, student info and departments provides several advantages over the string/dictionary representations in A1. Using enums to represent this data means that the data is immutable and will contain more information than their string/dictionary counterparts. For example, we are able to check to see if an object is of type GenT, DeptT, and SInfoT using the isinstance function, and check the type using the type function. It also prevents misspelled field names which can cause several issues such as causing the program to add or remove unintentionally. By using enums, it provides protection against problems with misspelling, capitalization etc.

E Code for StdntAllocTypes.py

```
from enum import Enum
from typing import NamedTuple

from SeqADT import *


class GenT(Enum):
    male = 0
    female = 1


class DeptT(Enum):
    civil = 0
    chemical = 1
    electrical = 2
    mechanical = 3
    software = 4
    materials = 5
    engphys = 6


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

F Code for SeqADT.py

```
## @file SeqADT.py
#  @author Donisius Wigie
#  @date 2019-02-09


## @brief This class stores the top choices of the students.
#  @details It will store a list of the DeptT top choices.
class SeqADT:
    ## @brief Constructor for the class SeqADT.
    #  @details Initiates the index for the choices list of top choices.
    def __init__(self, t):
        self.t = t
        self.i = 0

    ## @brief Sets the index back to 0
    def start(self):
        self.i = 0

    ## @brief Adds one to the index and returns the previous top choice.
    #  @return Returns the previous top choice before moving index.
    def next(self):
        temp = self.i
        if (self.i < self.t.__len__()):
            self.i += 1
        elif (self.i >= self.t.__len__()):
            raise Exception(StopIteration)
        return self.t[temp]

    ## @brief Checks to see if there are no more top choices.
    #  @return Returns True if there are no more choices and False if there is.
    def end(self):
        if(self.i >= self.t.__len__()):
            return True
        else:
            return False
```

G Code for DCapALst.py

```
## @file DCapLst.py
#  @author Donisius Wigie
#  @date 2019-02-09


## @brief This class represents the department capacities.
#  @details This class contains the capacities of each of the departments
class DCapALst:

    ## @brief Initiator for DCapLst
    #  @details Creates an empty list that will contain the departments and
    #  associated capacities.
    @staticmethod
    def init():
        DCapALst.s = []

    ## @brief Adds a department and the capacity
    #  @details Appends a tuple of the department and its capacity to DCapALst.s
    #  @param d DeptT containing the department name
    #  @param n Integer value of the capacity of the department
    @staticmethod
    def add(d, n):
        for i in range(DCapALst.s.__len__()):
            if(DCapALst.s[i][0] == d):
                raise Exception(KeyError)
        DCapALst.s.append((d, n))

    ## @brief Removes a department and its capacity from DCapLst.s
    #  @param d DeptT containing the department name
    @staticmethod
    def remove(d):
        for i in range(DCapALst.s.__len__()):
            if(DCapALst.s[i][0] is d):
                del DCapALst.s[i]
                return
        raise Exception(KeyError)

    ## @brief Checks to see if a department is in DCapLst.s
    #  @param d DeptT containing the department name
    #  @return Returns True if the department is in DCapLst.s
    @staticmethod
    def elm(d):
        for i in range(DCapALst.s.__len__()):
            if(DCapALst.s[i][0] is d):
                return True
        return False

    ## @brief Returns the capacity of a departments
    #  @param d DeptT containing the department name
    #  @return Returns the department d capacity
    @staticmethod
    def capacity(d):
        for i in range(DCapALst.s.__len__()):
            if(DCapALst.s[i][0] is d):
                return DCapALst.s[i][1]
        raise Exception(KeyError)
```

H Code for AALst.py

```
## @file AALst.py
#  @author Donisius Wigie
#  @date 2019-02-09
from StdntAllocTypes import *
## @brief This class represents the department allocations.
#  @details This class represents the departments and the list of students
#  who were allocated into each of the departments.


class AALst:

    ## @brief Initiator for AALst.
    #  @details Creates an empty dictionary with each department and an empty
    #  for each department where students can be allocated.
    @staticmethod
    def init():
        AALst.s = {DeptT.civil: [],
                   DeptT.chemical: [],
                   DeptT.electrical: [],
                   DeptT.mechanical: [],
                   DeptT.software: [],
                   DeptT.materials: [],
                   DeptT.engphys: []}

    ## @brief Adds a student to a department.
    #  @param dep DeptT containing the department name
    #  @param m String containing the macid of the student
    @staticmethod
    def add_stdnt(dep, m):
        AALst.s[dep].append(m)

    ## @brief Returns the list of students allocated into a department
    #  @param s DeptT containing the department name
    #  @return List of students allocated into a department
    @staticmethod
    def lst_alloc(d):
        return AALst.s[d]

    ## @brief Returns the number of students who are allocated in a specified department
    #  @param d DeptT containing the department name
    #  @return Integer value of number of students in a specified department
    @staticmethod
    def num_alloc(d):
        return sum(1 for i in AALst.s[d])
```

I Code for SALst.py

```
## @file SALst.py
#  @author Donisius Wigie
#  @date 2019-02-09
from StdntAllocTypes import *
from AALst import *
from DCapALst import *


## @brief This class represents the student allocation processes.
#  @details This class contains a list of the students with their info.
class SALst:

    ## @brief Initiator for SALst.
    #  @details Creates an empty list that will hold student info.
    @staticmethod
    def init():
        SALst.s = []

    ## @brief Function to add a student into SALst.
    #  @param m String containing the macID of a student.
    #  @param i SeqADT containing the information of the student.
    @staticmethod
    def add(m, i):
        if(isinstance(i, SInfoT)):
            SALst.s.append([m, i])
        else:
            raise Exception(KeyError)

    ## @brief Function to remove student and student information from SALst.
    #  @param m String containing the macID of a student.
    @staticmethod
    def remove(m):
        for i in range(SALst.s.__len__()):
            if(SALst.s[i][0] is m):
                del SALst.s[i]
                return
        raise Exception(KeyError)

    ## @brief Function to check if a student is present in SALst.
    #  @param m String containing the macID of a student.
    #  @return True if the student is in SALst and False if student is not in SALst.
    @staticmethod
    def elm(m):
        for i in range(SALst.s.__len__()):
            if(SALst.s[i][0] is m):
                return True
        return False

    ## @brief Function that returns a student's information
    #  @param m String containing the macID of a student.
    #  @return SeqADT containing the information of the student.
    @staticmethod
    def info(m):
        for i in range(SALst.s.__len__()):
            if(SALst.s[i][0] is m):
                return SALst.s[i][1]
        raise Exception(KeyError)

    ## @brief Function which sorts students according to gpa in descending order.
    #  @param f Function that filters which students will be returned in the list of students.
    #  @return Returns a list of stduents in descending order based on gpa, filtered by f.
    @staticmethod
    def sort(f):
        seq_str = []
        for i in range(SALst.s.__len__()):
            for j in range(i + 1, SALst.s.__len__()):
                if(SALst.s[i][1].gpa < SALst.s[j][1].gpa):
                    SALst.s[i], SALst.s[j] = SALst.s[j], SALst.s[i]
        for i in range(SALst.s.__len__()):
            if(f(SALst.s[i][1]) is True):
                seq_str.append(SALst.s[i][0])
        return seq_str

    ## @brief Function that gets the gpa average of all males or all females.
    #  @param f GenT gender of the students.
    #  @return Float value which is the gpa of all males or all females.
```

```
@staticmethod
def average(f):
    total = 0
    num_fset = 0
    for i in range(SALst.s.__len__()):
        if(f(SALst.s[i][1])):
            total += SALst.s[i][1].gpa
            num_fset += 1
    if(num_fset != 0):
        return float(total / num_fset)
    else:
        raise Exception('Runtime Error')

## @brief Function that allocates students into departments.
#  @details Sorts students who have freechoice before those who dont.
#  @details Students who are not put into their first choice get put to next choices.
@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)
    stud = SALst.sort(lambda t: not(t.freechoice) and t.gpa >= 4.0)
    for m in stud:
        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 Exception(RuntimeError)

@staticmethod
def get_gpa(m, s):
    for i in range(s.__len__()):
        if(m == s[i][0]):
            return s[i][1].gpa
```

J Code for Read.py

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


def load_stdnt_data(s):

    d = {'civil': DeptT.civil,
         'chemical': DeptT.chemical,
         'electrical': DeptT.electrical,
         'mechanical': DeptT.mechanical,
         'software': DeptT.software,
         'materials': DeptT.materials,
         'engphys': DeptT.engphys}
    SALst.init()
    file = open(s, "r")
    file.seek(0)
    number_of_lines = sum(1 for line in file)
    file.seek(0)
    for i in range(number_of_lines):
        temp = file.readline()
        temp2 = temp.strip().split(", ")
        temp2[5] = temp2[5][1:]
        temp2[-2] = temp2[-2][:-1]
        if(temp[3] == 'male'):
            gender = GenT.male
        else:
            gender = GenT.female
        top_choices = []
        for i in temp2[5:-1]:
            top_choices.append(d[i])
        free_choice = ''
        if(temp2[-1] == 'True'):
            free_choice = True
        else:
            free_choice = False
        sinfo = SInfoT(temp2[1], temp2[2], gender,
                       float(temp2[4]), SeqADT(top_choices), free_choice)
        SALst.add(temp2[0], sinfo)


def load_dcap_data(s):

    d = {'civil': DeptT.civil,
         'chemical': DeptT.chemical,
         'electrical': DeptT.electrical,
         'mechanical': DeptT.mechanical,
         'software': DeptT.software,
         'materials': DeptT.materials,
         'engphys': DeptT.engphys}
    DCapALst.init()
    file = open(s, "r")
    file.seek(0)
    number_of_lines = sum(1 for line in file)
    dept = ''
    file.seek(0)
    for i in range(number_of_lines):
        temp = file.readline()
        temp2 = temp.strip().split(", ")
        dept = d[temp2[0]]
        DCapALst.add(dept, int(temp2[1]))
```

K Code for Partner's SeqADT.py

```
## @file SeqADT.py
#  @title SeqADT
#  @author Jordan Levy 400131067
#  @date February 11th, 2019


## @brief This class is a Sequence ADT
#  @details This class represents a sequence that can only be indexed by the next element
class SeqADT:

    ## @brief This is a constructor for SeqADT
    #  @param x a sequence
    #  @return A new SeqADT object
    def __init__(self, x):
        self.s = x
        self.i = 0

    ## @brief This method resets the iteration back to the beginning of the sequence
    def start(self):
        self.i = 0

    ## @brief This method gets the next element in the sequence
    #  @throws StopIteration if there is no next element
    #  @return The next element in the sequence s
    def next(self):
        if(self.end()):
            raise StopIteration
        self.i += 1
        return self.s[self.i - 1]

    ## @brief This method checks if we have reached the end of the sequence
    #  @return A boolean, true if the end of the sequence has been reached, false if not
    def end(self):
        return self.i >= len(self.s)
```

L Code for Partner's DCapALst.py

```
## @file DCapALst.py
#  @title DCapALst
#  @author Jordan Levy 400131067
#  @date February 11th, 2019


## @brief This class is the Department Capacity Association List Module
#  @details This class represents the list of department capacities
class DCapALst:

    s = []

    ## @brief This method initializes the set s
    @staticmethod
    def init():
        DCapALst.s = []

    ## @brief This method inserts (d, n) into s if (d, ?) is not already in s
    #  @param d a key
    #  @param n a natural number value
    #  @throws KeyError if an element (d, ?) is already contained in the set s
    @staticmethod
    def add(d, n):
        for i in DCapALst.s:
            if i[0] == d:
                raise KeyError
        DCapALst.s.append([d, n])

    ## @brief This method removes the tuple (d, n) from the set s if it is in set s
    #  @param d a key
    #  @throws KeyError if and element (d, n) is not contained in the set s
    @staticmethod
    def remove(d):
        for i in DCapALst.s:
            if i[0] == d:
                DCapALst.s.remove(i)
                return
        raise KeyError

    ## @brief This method checks if an element (d, n) is in the set s
    #  @param d a key
    #  @return A boolean value, true if (d, n) is in set s, false if not
    @staticmethod
    def elm(d):
        for i in DCapALst.s:
            if i[0] == d:
                return True
        return False

    ## @brief This method returns the value n associated with (d, n) in set s
    #  @param d a key
    #  @throws KeyError if an element (d, n) is not contained in the set s
    #  @return A natural number, n, if (d, n) is in the set s
    @staticmethod
    def capacity(d):
        for i in DCapALst.s:
            if i[0] == d:
                return i[1]
        raise KeyError
```

M Code for Partner's SALst.py

```
## @file SALst.py
#  @title SALst
#  @author Jordan Levy 400131067
#  @date February 11th, 2019
from StdntAllocTypes import *
from AALst import *
from DCapALst import *


## @brief This class is the Student Association List Module
#  @details This class is made to allocate students to departments
class SALst:

    s = []

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

    ## @brief This method adds a student to the list of students s
    #  @param m the student's macid
    #  @param i the student's info, as an SInfoT type
    @staticmethod
    def add(m, i):
        for n in SALst.s:
            if n[0] == m:
                raise KeyError
        SALst.s.append([m, i])

    ## @brief This method removes a student from the list of students s
    #  @param m the student's macid
    #  @throws KeyError if no student with macid m exists in the list s
    @staticmethod
    def remove(m):
        for i in SALst.s:
            if i[0] == m:
                SALst.s.remove(i)
                return
        raise KeyError

    ## @brief This method checks if an element (m, i) is in the list s
    #  @param m a key
    #  @return A boolean value, true if (m, i) is in list s, false if not
    @staticmethod
    def elm(m):
        for i in SALst.s:
            if i[0] == m:
                return True
        return False

    ## @brief This method gets a student's info from their macid m
    #  @param m the student's macid
    #  @return the student's info, as an SInfoT type
    #  @throws KeyError if not student with macid m exists in the list s
    @staticmethod
    def info(m):
        for i in SALst.s:
            if i[0] == m:
                return i[1]
        raise KeyError

    ## @brief This method sorts the students in descending order by gpa
    #  @param f a filtering function
    #  @return the sorted list of students
    @staticmethod
    def sort(f):
        ret = SALst.s
        ret = list(filter(lambda x: f(x[1]), ret))  # apply filter function
        ret = sorted(ret, key=lambda x: x[1].gpa, reverse=True)  # sort the list by gpa
        ret = list(map(lambda x: x[0], ret))  # map the list to macid strings
        return ret

    ## @brief This method calculates the average gpa for students
    #  @param f a filtering function
    #  @return a real number representing the average gpa
```

```
@staticmethod
def average(f):
    ret = 0
    num_elements = 0
    for x in SALst.s:
        # check filter function
        if f(x[1]):
            ret += x[1].gpa
            num_elements += 1
    return ret / num_elements

## @brief This method allocates students to departments based on freechoice and gpa
#  @throws RuntimeError if a student was unable to be allocated to any department
@staticmethod
def allocate():
    AALst.init()
    free_choice_sort = SALst.sort(lambda t: t.freechoice and t.gpa >= 4.0)
    for m in free_choice_sort:
        ch = SALst.info(m).choices
        AALst.add_stdnt(ch.next(), m)

    non_free_choice_sort = SALst.sort(lambda t: not t.freechoice and t.gpa >= 4.0)
    for m in non_free_choice_sort:
        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 This method gets a student's gpa from their macid
#  @param m the student's macid
#  @param s the list of students
#  @return a real number representing the gpa of this student
def get_gpa(self, m, s):
    for n in s:
        if n[0] == m:
            return n[1].gpa
    return -1
```

N Code for test_All.py

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


# Tests for DCapALst
DCapALst.init()


# Tests to see if elm will return false is a department is not in DCapALst
def test_elm_empty():
    assert DCapALst.elm(DeptT.chemical) is False


# Tests if add works and if capacity works when capacity is 0
def test_capacity_empty():
    DCapALst.add(DeptT.chemical, 0)
    assert DCapALst.capacity(DeptT.chemical) == 0


# Tests to see if remove function works and if capacity will raise an exception if
# department info is not present
def test_remove_capacity():
    DCapALst.remove(DeptT.chemical)
    with pytest.raises(Exception):
        DCapALst.capacity(DeptT.chemical)


# Tests to see if add works for normal cases and if elm will work for normal cases
def test_add_elm():
    DCapALst.add(DeptT.software, 50)
    assert DCapALst.elm(DeptT.software) is True


def test_add_except():
    with pytest.raises(Exception):
        DCapALst.add(DeptT.software, 0)


# Tests for SALst
SALst.init()
DCapALst.init()


def test_remove_nothing():
    with pytest.raises(Exception):
        SALst.remove('wigied')


def test_add():
    sinfo1 = SInfoT("first", "last", GenT.male, 12.0,
                    SeqADT([DeptT.civil, DeptT.chemical]), True)
    SALst.add('wigied', sinfo1)
    assert SALst.elm('wigied') is True


def test_remove_elm():
    SALst.remove('wigied')
    assert SALst.elm('wigied') is False


def test_remove_nonexist():
    with pytest.raises(Exception):
        SALst.remove('pintos')


def test_info():
    sinfo1 = SInfoT("first", "last", GenT.male, 12.0,
                    SeqADT([DeptT.civil, DeptT.chemical]), True)
    SALst.add('wigied', sinfo1)
    assert isinstance(SALst.info('wigied'), SInfoT)
    assert SALst.info('wigied').gender == GenT.male
    assert SALst.info('wigied').fname == "first"
```

```
    SALst.remove('wigied')


def test_sort():
    sinfo2 = SInfoT("first", "last", GenT.male, 11.0,
                    SeqADT([DeptT.software, DeptT.chemical]), True)
    sinfo3 = SInfoT("first", "last", GenT.male, 10.0,
                    SeqADT([DeptT.electrical, DeptT.chemical]), True)
    sinfo4 = SInfoT("first", "last", GenT.male, 6.0,
                    SeqADT([DeptT.engphys, DeptT.chemical]), True)
    sinfo5 = SInfoT("first", "last", GenT.male, 3.0,
                    SeqADT([DeptT.chemical, DeptT.civil]), True)
    sinfo6 = SInfoT("first", "last", GenT.male, 12.0,
                    SeqADT([DeptT.mechanical, DeptT.chemical]), False)
    SALst.add('wigied2', sinfo2)
    SALst.add('wigied3', sinfo3)
    SALst.add('wigied4', sinfo4)
    SALst.add('wigied5', sinfo5)
    SALst.add('wigied6', sinfo6)
    sort_list = SALst.sort(lambda t: t.freechoice and t.gpa >= 4.0)
    assert sort_list == ['wigied2', 'wigied3', 'wigied4']


def test_sort_notfc():
    sort_list = SALst.sort(lambda t: not(t.freechoice) and t.gpa >= 4.0)
    assert sort_list == ['wigied6']


def test_average():
    average_males = SALst.average(lambda x: x.gender == GenT.male)
    assert average_males == 8.4


def test_average_excep():
    with pytest.raises(Exception):
        SALst.average(lambda x: x.gender == GenT.female)


def test_allocate_no_deps():
    with pytest.raises(Exception):
        SALst.allocate()
    SALst.remove('wigied2')
    SALst.remove('wigied3')
    SALst.remove('wigied4')
    SALst.remove('wigied5')
    SALst.remove('wigied6')


def test_allocate():
    sinfo2 = SInfoT("first", "last", GenT.male, 11.0,
                    SeqADT([DeptT.software, DeptT.chemical]), True)
    sinfo3 = SInfoT("first", "last", GenT.male, 10.0,
                    SeqADT([DeptT.electrical, DeptT.chemical]), True)
    sinfo4 = SInfoT("first", "last", GenT.male, 6.0,
                    SeqADT([DeptT.engphys, DeptT.chemical]), True)
    sinfo5 = SInfoT("first", "last", GenT.male, 3.0,
                    SeqADT([DeptT.chemical, DeptT.civil]), True)
    sinfo6 = SInfoT("first", "last", GenT.male, 12.0,
                    SeqADT([DeptT.mechanical, DeptT.chemical]), False)
    SALst.add('wigied2', sinfo2)
    SALst.add('wigied3', sinfo3)
    SALst.add('wigied4', sinfo4)
    SALst.add('wigied5', sinfo5)
    SALst.add('wigied6', sinfo6)
    DCapALst.add(DeptT.civil, 5)
    DCapALst.add(DeptT.electrical, 5)
    DCapALst.add(DeptT.engphys, 1)
    DCapALst.add(DeptT.chemical, 5)
    DCapALst.add(DeptT.mechanical, 0)
    SALst.allocate()
    assert AALst.lst_alloc(DeptT.chemical) == ['wigied6']


def test_allocate_no_space():
    sinfo7 = SInfoT("first", "last", GenT.male, 10.0,
                    SeqADT([DeptT.mechanical]), False)
    SALst.add('wigied7', sinfo7)
    with pytest.raises(Exception):
        SALst.allocate()
```

```
# Tests for SeqADT
def test_constructor():
    constructor_test = SeqADT([DeptT.software, DeptT.electrical, DeptT.engphys])
    assert isinstance(constructor_test, SeqADT)


def test_next():
    constructor_test = SeqADT([DeptT.software, DeptT.electrical, DeptT.engphys])
    next_test = constructor_test.next()
    assert next_test is DeptT.software


def test_next_except():
    constructor_test = SeqADT([DeptT.software, DeptT.electrical, DeptT.engphys])
    constructor_test.next()
    constructor_test.next()
    constructor_test.next()
    with pytest.raises(Exception):
        constructor_test.next()


def test_end():
    constructor_test = SeqADT([DeptT.software, DeptT.electrical, DeptT.engphys])
    assert constructor_test.end() is False


def test_end_2():
    constructor_test = SeqADT([DeptT.software, DeptT.electrical, DeptT.engphys])
    constructor_test.next()
    constructor_test.next()
    constructor_test.next()
    assert constructor_test.end() is True
```