Assignment 2 Solution

Joshua Guinness, guinnesj, 400134735

February 25, 2019

In this assignment, a program was created to place students into their second year engineering stream based on their GPA, free choice, and preference of stream. This assignment was implemented by following a formal specification. The results of testing, comments on the specification, as well as answers to other questions are in the below sections.

1 Testing of the Original Program

The approach to testing that was taken was to first test the basic functionality of each function in each module given the input data in the text files to ensure everything worked. Then the more boundary and weird cases would be tested. Because of the time constraint, only 15 test cases were created instead of the high amount that were planned. Thus, the test cases that were implemented, focused on ensuring the basic functionality of the program and checking to see whether key functions in certain modules worked correctly and behaved as specified. All 15 test cases passed. There were no problems uncovered while testing but that is likely because the testing was not thourough enough to uncover potential errors.

2 Results of Testing Partner's Code

After running the test cases on the partner's code, 11 of the 15 test cases passed and 4 failed. Three of the four that failed were just assertions checking the department capacity immediately after reading in the file of department capacities. The reason these failed is because the value returned by the partners code is a string, while it is comparing it to an int. This is an error in the partner implementation. The fourth test case failed due to an error in their allocate() function in the SALst module. In their code, they tried to see whether an int value was less than a string, an operation obviously not supported therefore raising a TypeError. This is again an error in the implementation of their code. However, the 11 that did pass did so because of correct implementation due to following the specification. Having more test cases may have exposed more errors in implementation.

3 Critique of Given Design Specification

This specification was a lot more formal and had a lot less flexibility than the previous specification for Assignment 1. This had both disadvantages and advantages.

The main advantage is that the project structure, and implementation were crystal clear, leaving little room for misinterpretation or assumptions. As a programmer, not having to make assumptions makes coding much easier because certain things don't have to be accounted for e.g. bad input, and one spends less time thinking about the different assumptions one is making and whether they are valid or not. In terms of the implementation being specified, this is really advantageous because it separates the coding from the planning, meaning when your coding your coding, and when your planning your planning. It is easy to fall into the trap of planning while coding but this is bad because there is a lack of direction and organization and time ends up being wasted by constantly changing it. The formal specification helps enforce the idea of planning before coding, making the time spent coding much more productive. One final advantage is that although the specification was very formal about how the modules are laid out and the exact functions in each module, it left how the functions are to be coded relatively open, allowing each programmer to code in their own style.

There are two main disadvantages of the specification that was given in this assignment. The first is that the formal language can often be hard to read and thus interpret. The high level of formality was occasionally difficult to understand what exactly was supposed to be changing, or happening, and in what circumstance. An example of this is the output specification for the sort function in the SALst module. A second disadvantage is that there is no high level overview of how all the modules are supposed to work together, rather it is left to the reader/programmer to figure out. This can be quite difficult and take a while and often leads to programming and implementing the modules without understanding how they work together.

Based on the comments above, two design changes that would benefit the intended audience of the specification are to give a high level overview of how all the modules work together, and english explanations to complicated formal specification statements. The first of these suggestions will help the reader understand the over flow and organization of the program without having to read the formal specification multiple times. This will give the programmer a better idea of how things work together and allow them to get started quicker. The second suggestion, will make more clear long formal statements, again, allowing the programming to understand and implement quicker.

To conlude, the formal specification is a great tool that drastically decreases the number of assumptions being made and ambiguous statements. It also enforces planning before programming making the implementation easier. However, it can be harder to understand how modules work together and the meaning behind long statements which is why improvements should be made in that regard.

4 Answers

1. Contrast the natural language of A1 to the formal specification of A2. What are the advantages and disadvantages of each approach?

   A few of the advantages and disadvantages were talked about in the previous section but they will be repeated here as well as added on to.

   The natural language of A1 made is easy to understand what the program was trying to accomplish, and how it was going to do that. It was really clear how the different modules were going to work together and depend on each other. The natural language also made it easy to understand what the purpose of each function would be. This is its main advantage, its easy of understand. Its disadvntage, however, is its ambiguity and openness. The lack of detail in the natural specification leads the reader confused exactly how things are supposed to be implemented, forcing many assumptions to be made. A further disadvantage is that the natural specification allows one to program without technically planning first, leading to wasted time, re-writing code, and planning while coding which is not a good idea.

   The main advantage of the formal specification of A2 is how clear everything is made and the fact that very few, if nothing is left to assumption or guesswork. It also further specified how to implement the code, leaving less planning in the hands of the programmer, leading to more productivity and a more accurate implementation. Its drawback lies in the fact that it can often be hard to read and observe how all the modules work together.

2. The specification makes the assumption that the gpa will be between 0 and 12. How would you modify the specification to change this assumption to an exception? Would you need to modify the specification to replace a record type with a new ADT?

   To modify the specification to change this assumption to an exception, it will just specify an exception that when a GPA that is not between 0 and 12 occurs, an exception is raised. A good place to implement this would be in the add method in SALst as this is when a student is added to a list of students. The GPA of the student can then be easily checked, raising an exception if the value is out of bounds. Another place this could possibly be done is in the Read module. Although the specification would not need to be changed to replace a record type with a new ADT, it would likely be better, and result in a cleaner implementation and would be easier to change in the future. With it implemented as an ADT, custom functions could also be added which would benefit the programs as well.

3. If we ignore the functions sort, average and allocate, the two modules SALst and DCapALst are very similar. Ignoring the functions mentioned, how could the documentation be modified to take advantage of the similarities?

   The documentation can be modified to take advantage of the similarities between SALst and DCapALst by reusing sections and just identifying which key elements or variables need to change between modules. To be more specific, the documentation for one function will be used for both functions, with some additional notes about what specifically in the function will change or be different between modules. This is a good idea because it will cut down on the amount of writing and show how similar things are, though it does have the downside of potentially being more confusing.

4. A1 had a question about generality of an interface. In what ways is A2 more general than A1?

   A2 is more general than A1 in that there is more freedom in the implementation within the functions. A1 was pretty specific in regards to exactly how the code in the functions should be written while A2 has more freedom in that sense. A2 also has less restrictions in the exact data type to use, choosing instead to just specify a tuple, but leaving it open how the tuple should be implemented. A1 was much more specific in this sense, detailing exactly the data type to use, which was a dictionary in most cases, with no leeway.

5. The list of choices for each student is represented by a custom object, SeqADT, instead of a Python list. For this specific usage, what are the advantages of using SeqADT over a regular list?

   An advantage of representing the list of choices for each student as a custom object, in this case SeqADT, is that custom operations can be defined on them, often making them easier to use. For this specific usage, SeqADT has the functions start, next, and end, which enable the client to use the object in different ways than it would be able to do with just a list. THe next() function is especially helpful because it provides iterating through the sequence much easier than trying to iterate through a list on the client side.

6. Many of the strings in A1 have been replaced by enums in A2. For these cases, what advantages do enums provide? Why weren't enums also introduced in the specification for macids?

   Enums provide the advantage in that there is only a specific set of values that can be chosen, massively reducing the margin of error. Enums are especially useful for constants. In the case for this assignment, enumerated types are used for gender and engineering departments. This means that students will have a guarenteed value that has meaning, this also means less error checking has to be in place when using enumerated types. Enums were not introduced in the specification for macids because macids are of an infinite variety, meaning they are variables. Macids cannot be pre-planned and therefore cannot be enums.

E Code for StdntAllocTypes.py

```
## @file StdntAllocTypes.py
#  @title StdntAllocTypes
#  @author Joshua Guinness, guinnesj, 400134735
#  @date Febuary 11, 2019

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

## @brief Enumeration class for gender
class GenT(Enum):
    male = 1
    female = 2

## @brief Enumeration class for engineering departments
class DeptT(Enum):
    civil = 1
    chemical = 2
    electrical = 3
    mechanical = 4
    software = 5
    materials = 6
    engphys = 7

## @brief NamedTuple for info about each student
class SInfoT(NamedTuple):
    fname: str
    lname: str
    gender: GenT
    gpa: float
    choices: SeqADT
    freechoice: bool
```

F Code for SeqADT.py

```
## @file SeqADT.py
#  @title SeqADT
#  @author Joshua Guinness, guinnesj, 400134735
#  @date Febuary 11, 2019

## @brief A class which defines an abstract data type which is a sequence
class SeqADT:

    ## @brief Initializ the state variables
    #  @param p2 A sequence of T.
    def __init__(self, x):
        self.s = x
        self.i = 0

    ## @brief Sets the integer variable to zero.
    def start(self):
        self.i = 0

    ## @brief Outputs the current element of the sequence then moving to the next element
    #  @return The current element of the sequence
    def next(self):
        if (self.i >= len(self.s)):
            raise StopIteration

        return self.s[self.i]
        self.i = self.i + 1

    ## @brief Checks to see if the end of the sequence is reached
    #  @return Boolean value about whether have reached end of sequence
    def end(self):
        if (self.i >= len(self.s)):
            return True
        else:
            return False
```

G Code for DCapALst.py

```python
## @file DCapALst.py
#  @title DCapALst
#  @author Joshua Guinness, guinnesj, 400134735
#  @date Febuary 11, 2019

from StdntAllocTypes import *


## @brief Departments and their capacities and functions to preform operations on them
class DCapALst:

    ## @brief Makes the list empty
    @staticmethod
    def init():
        DCapALst.s = []

    ## @brief Adds a department and its capacity to the list
    #  @param p1 Department of type DeptT
    #  @param p2 Capacity of type integer
    @staticmethod
    def add(d, n):

        if (len(DCapALst.s) == 0):
            DCapALst.s.append((d, n))
        else:
            is_inside = False
            for i in DCapALst.s:
                if (i[0] == d):
                    is_inside = True

            if (is_inside == True):
                raise KeyError
            else:
                DCapALst.s.append((d, n))

    ## @brief Removes a department from the set
    #  @param p1 Department of type DeptT
    @staticmethod
    def remove(d):
        is_inside = False
        for i in DCapALst.s:
            if (i[0] == d):
                is_inside = True
                DCapALst.s.remove(i)

        if (is_inside == False):
            raise KeyError

    ## @brief Checks to see if a department already exists in the set
    #  @param p1 Department of type DeptT
    #  @return Boolean value about whether the department already exists
    @staticmethod
    def elm(d):
        is_inside = False
        for i in DCapALst.s:
            if (i[0] == d):
                is_inside = True

        if (is_inside == True):
            return True
        else:
            return False

    ## @brief Checks the current capacity of a department
    #  @param p1 Department of type DeptT
    #  @return Capacity of the passed department
    @staticmethod
    def capacity(d):
        is_inside = False
        for i in DCapALst.s:
            if (i[0] == d):
                is_inside = True
                return int(i[1])

        if (is_inside == False):
            raise KeyError
```

H Code for AALst.py

```
## @file AALst.py
#  @title AALst
#  @author Joshua Guinness, guinnesj, 400134735
#  @date Febuary 11, 2019

from StdntAllocTypes import *


## @brief Departments and the students allocated to them
class AALst:

    ## @brief Makes the list empty
    @staticmethod
    def init():
        AALst.s = []

        for dept in (DeptT):
            AALst.s.append((dept, []))

    ## @brief Adds a student to a department
    #  @param p1 Department of type DeptT
    #  @param p2 MacId of student
    @staticmethod
    def add_stdnt(dep, m):
        for i in AALst.s:
            if (i[0] == dep):
                i[1].append(m)

    ## @brief Outputs a list of students allocated to the specified department
    #  @param p1 Department of type DeptT
    #  @return list of allocated macids to the specified department
    @staticmethod
    def lst_alloc(d):
        for i in AALst.s:
            if (i[0] == d):
                return i[1]

    ## @brief Checks the number of students allocated to a specified department
    #  @param p1 Department of type DeptT
    #  @return Number of students allocated to a specified department
    @staticmethod
    def num_alloc(d):
        for i in AALst.s:
            if (i[0] == d):
                return len(i[1])
```

I Code for SALst.py

```
## @file SALst.py
#  @title SALst
#  @author Joshua Guinness, guinnesj, 400134735
#  @date Febuary 11, 2019

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


## @brief Students and operations to preform on them
class SALst:

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

    ## @brief Adds a student to the list
    #  @param p1 macid of student
    #  @param p2 student info
    @staticmethod
    def add(m, i):
        is_inside = False
        for j in SALst.s:
            if (j[0] == m):
                is_inside = True

        if (is_inside == True):
            raise KeyError
        else:
            SALst.s.append((m, i))

    ## @brief Removes a student from the list
    #  @param p1 macid of a student
    @staticmethod
    def remove(m):
        is_inside = False
        for i in SALst.s:
            if (i[0] == m):
                is_inside = True
                SALst.s.remove(i)

        if (is_inside == False):
            raise KeyError

    ## @brief Checks to see if a student exists in the list
    #  @param p1 macid of a student
    #  @return Boolean about whether the student exists
    @staticmethod
    def elm(m):
        is_inside = False
        for i in SALst.s:
            if (i[0] == m):
                is_inside = True

        if (is_inside == True):
            return True
        else:
            return False

    ## @brief Gets the info about a particular student
    #  @param p1 macid of a student
    #  @return Info about the specified student
    @staticmethod
    def info(m):
        is_inside = False
        for i in SALst.s:
            if(i[0] == m):
                is_inside = True
                return i[1]

        if (is_inside == False):
            raise KeyError

    ## @brief Sorts the student in decreasing order of GPA
```

```
    #  @param p1 lamda function
    #  @return list of macids of sorted students
    @staticmethod
    def sort(f):
        l = []
        temp = SALst.s.copy()
        to_delete = []
        counter = 0
        for i in temp:
            if (f(i[1]) == False):
                to_delete.append(counter)
            counter = counter + 1

        to_delete.reverse()
        for i in to_delete:
            temp.remove(temp[i])

        while (len(temp) > 0):
            highest = -1
            element_number = 0
            for i in range(len(temp)):
                if (temp[i][1].gpa > highest):
                    temp[i][1].gpa > highest
                    element_number = i
                    i = i+1

            l.append(temp[element_number][0])
            temp.remove(temp[element_number])

        return l

    ## @brief Checks to see if the end of the sequence is reached
    #  @param p1 The instance of the class
    #  @return Boolean value about whether have reached end of sequence
    @staticmethod
    def average(f):

        temp = SALst.s.copy()

        to_delete = []
        counter = 0
        for i in temp:
            if (f(i[1]) == False):
                to_delete.append(counter)
            counter = counter + 1

        to_delete.reverse()
        for i in to_delete:
            temp.remove(temp[i])

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

        total = 0
        number = len(temp)

        for i in temp:
            total = total + i[1].gpa

        return total/number

    ## @brief Checks to see if the end of the sequence is reached
    #  @param p1 The instance of the class
    #  @return Boolean value about whether have reached end of sequence
    @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
#  @title Read
#  @author Joshua Guinness, guinnesj, 400134735
#  @date Febuary 11, 2019

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


## @brief Loads in the student data and updates the state of the SALst module
#  @param A filename of student data
def load_stdnt_data(s):

    SALst.init()

    f = open(s, 'r')

    for line in f:
        temp = line.split(', ')
        student_info = []
        student_info.append(temp[1])
        student_info.append(temp[2])
        student_info.append(GenT[temp[3]])
        student_info.append(float(temp[4]))

        list_dept = []

        for i in range(5, len(temp) - 1):
            temp2 = temp[i].replace('[', "")
            temp3 = temp2.replace(']', "")
            temp4 = DeptT[temp3]

            list_dept.append(temp4)

        student_info.append(SeqADT(list_dept))
        string = temp[-1]
        string2 = string.replace('\n', "")
        if (string2 == "True"):
            student_info.append(True)
        else:
            student_info.append(False)

        final_info = SInfoT(student_info[0], student_info[1], student_info[2],
        student_info[3], student_info[4], student_info[5])
        SALst.add(temp[0], final_info)

    f.close()


## @brief Loads in the department data and updates the state of the DCapALst module
#  @param p1 A filename of department data
def load_dcap_data(s):

    DCapALst.init()

    f = open(s, 'r')
    for line in f:
        string = line.rstrip('\n')
        temp = string.split(', ')
        DCapALst.add(DeptT[temp[0]], temp[1])

    f.close()
```

K Code for testAll.py

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


class TestingClass:

    def setup_method(self, method):
        load_dcap_data("src/DeptCap.txt")
        load_stdnt_data("src/StdntData.txt")

    ## Testing the DCapALst modle

    # Checking to see whether departments correctly exist after reading in the data
    def test_CivilExists(self):
        assert DCapALst.elm(DeptT.civil)

    def test_ChemExists(self):
        assert DCapALst.elm(DeptT.chemical)

    def test_ElecExists(self):
        assert DCapALst.elm(DeptT.electrical)

    def test_MechExists(self):
        assert DCapALst.elm(DeptT.mechanical)

    def test_SoftExists(self):
        assert DCapALst.elm(DeptT.software)

    def test_MatExists(self):
        assert DCapALst.elm(DeptT.materials)

    def test_Phys(self):
        assert DCapALst.elm(DeptT.engphys)

    # Checking to see if getting the current capacity of the department works

    def test_GetCapacityCivil(self):
        assert DCapALst.capacity(DeptT.civil) == 100

    def test_GetCapacityMech(self):
        assert DCapALst.capacity(DeptT.mechanical) == 100

    def test_GetCapacityPhys(self):
        assert DCapALst.capacity(DeptT.engphys) == 100

    # Checking to see if removing a department works

    def test_RemoveDept(self):
        DCapALst.remove(DeptT.software)
        assert not DCapALst.elm(DeptT.software)

    # Checking to see if adding a department works

    def test_AddDept(self):
        DCapALst.remove(DeptT.software)
        DCapALst.add(DeptT.software, 100)
        assert DCapALst.elm(DeptT.software)

    # Checking the SALst module

    def test_StudentExists(self):
        assert SALst.elm("brownc")

    def test_RemoveStudent(self):
        SALst.remove("brownc")
        assert not SALst.elm("brownc")

    # Checking the AALst module

    def test_StudentAdd(self):
        SALst.allocate()
        AALst.add_stdnt(DeptT.civil, "brownc")
```

```
assert AALst.num_alloc(DeptT.civil) == 2
```

L Code for Partner's SeqADT.py

```
## @file SeqADT.py
#  @author Michael Barreiros
#  @brief SeqADT
#  @date 09/02/2019

## @brief An abstract data type for a sequence


class SeqADT:

    s = []
    i = 0
    ## @brief SeqADT constructor
    #  @details initalizes the sequence with a given sequence
    #  @param x is a sequence of type T that SeqADT will be initialized to
    #  @return returns itself, a SeqADT type
    def __init__(self, x):
        self.s = x
        self.i = 0

    ## @brief start method
    #  @details resets the iterator i to 0, which is the "start" of the
    #  sequence
    def start(self):
        self.i = 0

    ## @brief next method
    #  @details returns the sequence at i and adds one to the iterator, this
    #  effectively moves the iterator to the next element in the sequence
    #  @exception throws StopIteration if i is greater or equal to the
    #  size of s
    #  @return returns s[i] before i got one added to it
    def next(self):
        if self.i >= len(self.s):
            raise StopIteration
        temp = self.s[self.i]
        self.i = self.i + 1

        return temp

    ## @brief end method
    #  @details this function's purpose is to return whether or not i is
    #  at the end of s
    def end(self):
        return self.i >= len(self.s)
```

M Code for Partner's DCapALst.py

```
## @file DCapALst.py
#  @author Michael Barreiros
#  @brief DCapALst
#  @date 09/02/2019

# from StdntAllocTypes import GenT, DeptT, SInfoT

## @brief DCapALst is an abstract data dype


class DCapALst:
    s = {}

    ## @brief the constructor for DCapALst
    #  @details sets the sequence to be an empty sequence
    @staticmethod
    def init():
        DCapALst.s = {}

    ## @brief the elm function
    #  @details returns whether or not a department is an element
    #  of the sequence
    #  @param d the department name
    #  @return a boolean value of whether or not the department is
    #  in the sequence
    @staticmethod
    def elm(d):
        return d in DCapALst.s

    ## @brief the add function
    #  @details adds a department and its capacity to the sequence
    #  @param d the department name
    #  @param n the department capacity
    #  @exception KeyError if d is already in the sequence
    @staticmethod
    def add(d, n):
        if DCapALst.elm(d):
            raise KeyError
        DCapALst.s[d] = n

    ## @brief the remove function
    #  @details removes a department and its capacity value from the sequence
    #  @param d the department name
    #  @exception KeyEror if d is not in the sequence
    @staticmethod
    def remove(d):
        if not(DCapALst.elm(d)):
            raise KeyError
        del DCapALst.s[d]

    ## @brief the capacity function
    #  @details outputs the capacity value of a given department
    #  @param d the department name
    #  @exception KeyError if d is not in the sequence
    #  @return DCapALst.s[d] this is the capacity of the department
    #  that was given
    @staticmethod
    def capacity(d):
        if not(DCapALst.elm(d)):
            raise KeyError

        return DCapALst.s[d]
```

N Code for Partner's SALst.py

```
## @file SALst.py
#  @author Michael Barreiros
#  @brief SALst
#  @date 11/02/2019

# from StdntAllocTypes import GenT, DeptT, SInfoT
from AALst import AALst
from DCapALst import DCapALst

## @brief SALst an abstract data type for an allocated list of students


class SALst:

    s = {}

    ## @brief the constructor for SALst
    @staticmethod
    def init():
        SALst.s = {}

    ## @brief the elm function
    #  @details returns a boolean for whether or not m exists in the set
    #  @return a boolean value for whether or not m exists in the set
    @staticmethod
    def elm(m):
        return m in SALst.s

    ## @brief the add function
    #  @details adds a student by their macid m to the list
    #  @param m the student's macid
    #  @param i the student info of type SInfoT associated with the student
    #  @exception KeyError if the macid m already appears in the set
    @staticmethod
    def add(m, i):
        if SALst.elm(m):
            raise KeyError
        SALst.s[m] = i

    ## @brief the remove functtion
    #  @details removes a student by their macid m from the set
    #  @param m the student's macid
    #  @exception KeyError if the macid is not in the set
    @staticmethod
    def remove(m):
        if not(SALst.elm(m)):
            raise KeyError
        del SALst.s[m]

    ## @brief the info function
    #  @details this function returns the Student information for a given macid
    #  @param m the student's macid
    #  @exception KeyError if the given student doesn't exist in the set
    #  @return the student information of type SInfoT
    @staticmethod
    def info(m):
        if not(SALst.elm(m)):
            raise KeyError

        return SALst.s[m]

    ## @brief the sort function
    #  @details sorts all members of the set that are filtered by a function f
    #  @param f a function to be applied to the sequence. It takes aspects of
    #  SInfoT and returns a boolean
    #  @return L a sequence of strings that are sorted based on the function
    #  that was passed through
    @staticmethod
    def sort(f):
        usrtd = {}
        for macid in SALst.s:
            if f(SALst.info(macid)):
                usrtd[macid] = SALst.info(macid)
        ## newList was sorted using a line of code that was found
        #  on stackoverflow
        #  link is https://stackoverflow.com/questions/72899/
```

```python
    # how-do-i-sort-a-list-of-dictionaries-by-a-value-of-the-dictionary
    srtd = sorted(usrtd, key=lambda k: SALst.info(k).gpa, reverse=True)

    return srtd

## @brief the average function
#  @details computes the average following a criteria given through the
#  function file
#  @param f a function that filters the set
#  @exception ValueError if fset is an empty set which would cause
#  a division by zero
#  @return a float value for the average
@staticmethod
def average(f):
    fset = {}
    accumulated_gpa = 0
    for macid in SALst.s:
        if f(SALst.info(macid)):
            fset[macid] = SALst.info(macid)
            accumulated_gpa = accumulated_gpa + SALst.info(macid).gpa

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

    return accumulated_gpa / len(fset)

## @brief the allocate function
#  @details sorts freechoice students and other students then allocates
#  freechoice students first and then allocates the other students
#  @exception throws Runtimeerror if a student does not get allocated
@staticmethod
def allocate():
    AALst.init()
    freechoice_stdnts = SALst.sort(lambda t: t.freechoice and t.gpa >= 4.0)
    other_stdnts = SALst.sort(lambda t: not(t.freechoice) and t.gpa >= 4.0)

    for macid in freechoice_stdnts:
        choices = SALst.info(macid).choices
        AALst.add_stdnt(choices.next(), macid)

    for macid in other_stdnts:
        choices = SALst.info(macid).choices
        allocated = False
        while(not(allocated) and not(choices.end())):
            dept = choices.next()
            if AALst.num_alloc(dept) < DCapALst.capacity(dept):
                AALst.add_stdnt(dept, macid)
                allocated = True
        if not(allocated):
            raise RuntimeError
```