Assignment 2 (Individual Assignment)

Due Date

22 April 2023 17:00 (Saturday)

Declaration of Original Work

Plagiarism is a serious misconduct. No part of students' assignment should be taken from other people's work without giving them credit. All references must be clearly cited. Any plagiarism found in students' completed assignments can lead to disciplinary actions such as mark deduction, disqualification or even expulsion by the College.

By submitting this assignment to the subject lecturer through Moodle, you hereby declare that the work in this assignment is **completely** your own work. No part of this assignment is taken from other people's work without giving them credit. All references have been clearly cited.

You understand that an infringement of this declaration leaves you subject to disciplinary actions such as mark deduction, disqualification or even expulsion by the College.

Instruction

You are required to submit a Python Notebook (with the template provided, **Assign_2_Template.ipynb**) showing all the answers and programs and a **PDF file** (.pdf) showing **ALL** the executed output of the Python Notebook with **ALL** programs are executed successfully. The ipynb and PDF should be the **same version**, otherwise 50% of the total marks of that question will be deducted.

Your Python Notebook file should contain your name, your student ID no, course code (SEHH2239), course name (Data Structures), class (201) and date.

All programs must be written in Python programming language. For each programming-type question, please also write down the above-mentioned particulars as comments on source file, compile and test it. Each program must be successfully compiled and can execute, otherwise 50% of the total marks of that question will be deducted.

All submitted assessments will be evaluated with **Python version 3.8** (the current python version used in Google Colab). Your submitted assessments must run without errors on **Google Colab**.

Unless otherwise instructed, you **MUST NOT** import any modules in your submitted assessments.

You MUST NOT change the procedure name (include cases) and parameters required.

Items to be Submitted

To download the Python Notebook (.ipynb)

In Google Colab, File → Download → Download .ipynb

To download the PDF (.pdf)

In Google Colab, execute all the codes (Ctrl+F9),
File → Print → Printer choose "Micrpsoft Print to PDF"

1. Rename the Python Notebook and PDF file to with Student Name_Student ID No e.g. ***ChanTaiMan_21001234A.ipynb, ChanTaiMan_21001234A.pdf***
2. Put the Python Notebook (.ipynb) file together with the PDF file (.pdf) into a folder (**folder name format:** Student Name_Student ID No e.g**.** ChanTaiMan_21001234A). Use a compression software (e.g. Winzip) to compress the folder (e.g. ChanTaiMan_21001234A.zip) (***Submit the compressed folder e.g. ChanTaiMan_21001234A.zip via centralized class Moodle. In case the Moodle's Assignment Submission System is not available, send the softcopy to subject lecturer through e-mail***)

Attention:

While submitting the **softcopies** via Moodle, a timestamp will be placed on the softcopies of your assignment. There will be a sharp cut-off time at Moodle, so late assignments will be recorded at Moodle. Softcopies submitted via email or other means will NOT be accepted unless the Moodle is not available. As many students will submit their assignments to Moodle at around the deadline time, it normally takes longer for uploading your assignment, so it is strongly suggested that you start submitting earlier, say at least 45 minutes before the deadline. Marks will be deducted for late submission. Successful submission of this assignment includes the submission of both **ipynb** and **PDF**. Missing either **ipynb** or **PDF** is not successful submission.

Plagiarism will be penalized severely. Marks will be deducted for assignments that are plagiarized in whole or in part, regardless of the sources.

Late submission is liable to a **penalty of 10%** of the available marks for **each day late**; Saturdays, Sundays and holidays are counted. **Submission after 29 April 2023 17:00 will not be accepted**.

Question 1

A *stack* is a linear data structure that can be accessed only at one of its ends for storing and retrieving data. New elements are put on the top of the stack and taken off the top. The last element put on the stack is the first element removed from the stack. For this reason, a *stack* is called last in/first out (LIFO) structure.

a) Complete the following class `Stack` according to the comment described:

```
class Stack:
  def __init__(self) :
        # implement a stack using list

  def clear(self):
        # clear the stack

  def isEmpty(self):
        # Check if the stack is empty
        # return True if the stack is empty
        # otherwise return False

  def push(self, data):
        # Put data on the top of the stack

  def pop(self):
        # Take the topmost data from the stack and return it
        # if the stack is empty, return None

  def peek(self):
        # Return the topmost data in the stack
        # without removing it
        # if the stack is empty, return None

  def display(self):
        # show all the data in the stack
```

b) Given a stack of elements, we can sort it in ascending order using another temporary stack. Define a class `StackSort` which use *stack* to sort a list of `Student` according to their score, complete coding as the comment described. The class `Student` is provided below:

```
class Student:
  def __init__(self,name,score):
        self.name = name
        self.score = score
  def __str__(self):
        return f"{self.name}:{self.score}"
  def __repr__(self):
        return str(self)
  def __eq__(self,other):
        return (self.score == other.score
              and self.name == other.name)
  def __gt__(self,other):
        if (self.score == other.score):
            return self.name > other.name
        else:
            return self.score > other.score
  def __lt__(self,other):
        if (self.score == other.score):
            return self.name < other.name
        else:
            return self.score < other.score
  def __ge__(self,other):
        return (not self < other)
  def __le__(self,other):
        return (not self > other)

class StackSort:
  def __init__(self,data):
        # define two Stack
        # the first Stack contains all the elements in data
# and the second Stack left empty

  def sort(self):
# sort the elements in the first Stack as follows:
        # While the first Stack is not empty, do this:
        # - Pop an element from the first Stack and call it temp
        # - While the second Stack is not empty and
        #   top of the second stack is greater than temp:
        #     - Pop from the second Stack and
        #       push it to the first Stack
        # - Push temp to the second Stack
        # the sorted elements are in second Stack

  def display(self):
        # show the data in two Stacks
```

c) Execute the following program and show the output.

```
data = [Student("May",85),Student("Peter",30),Student("Louis",90),
Student("Frankie",30),Student("Ron",74)]
stackSort = StackSort(data)
print("Before stack sort:")
stackSort.display()
stackSort.sort()
print("After stack sort:")
stackSort.display()
```

Please provide **THREE** testing cases for the program.

d) What is the time complexity of the above stack sort?

Question 2

Given the following hash table with the hash function

```
h(key) = key mod 9
```

(mod means calculating the remainder. For example: 8 mod 3 = 2)

Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Value									

The keys *a* to *g* are formed by your student number with following form.

Keys	Based on your student ID number, formed by
a	4th and 5th digits
b	5th and 6th digits
c	6th and 7th digits
d	7th and 8th digits
e	4th and 6th digits
f	5th and 7th digits
g	6th and 8th digits

a) Insert the keys *a* to *g* into an empty hash table in sequential order using separate chaining as collision resolution technique. Draw the result table.

b) Repeat (a) by using linear probing as collision resolution technique. Draw the result table.

c) When deleting a key from a hash table using linear probing, can we simply make the slot as empty? Explain by using your answer in b) as example and suggest a way that can avoid the problem (if any).

Question 3

The aim of Questions 3 and 4 is to build a data structure for storing and fast retrieving students' course enrollment records. The data structure combines various basic data structures (provided in Assignment Template) taught in classes. The main task is therefore an integration of several existing data structures to put them into a higher-level use. In Question 3, you will be asked to build a data structure for storing records of students registered in a school. To facilitate a fast searching, a Binary Search Tree (BST) will be used to organize objects of the class `Student`. Student IDs (in the format of 8 digits followed by the alphabet "A") will be used as the search key. To further speed up the process, a list of 10 BSTs will be used, where the `i`-th BST stores the information about students whose last digit of their student IDs are `i` (for `i = 0, 1, …, 9`). Thus, it has a flavor of hashing too. The data structure for Question 3 is depicted in Figure 1, where a list of 10 BSTs and the structure of each BST are shown.

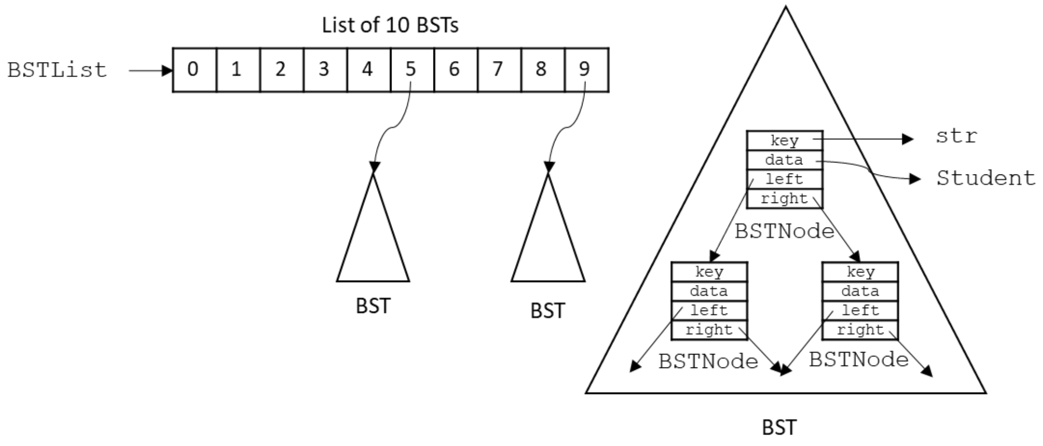

Figure 1.

(a) Write a class `Student` with the following specification:
   1) The constructor takes two inputs: `name` (type `str`) and `studentid` (type `str`). Validate their types.
   2) Validate the input `studentid` using the Boolean value "`re.fullmatch('[0-9]{8}A', studentid) is None`" (a `True` value designates a mismatch). It checks whether the input has 8 digits followed by the character `'A'`. You need to import the `re` library. Raise an error and stop the program if the input is invalid.
   3) Initialize an instance attribute `name` to the input `name`.
   4) Initialize an instance attribute `studentid` to the input `studentid`.
   5) Overload the `__str__()` method to facilitate the printing with `print()`. See lines 2 & 3 of the outputs below for the required format.
   6) Overload the `__eq__()` method to facilitate the comparison of `Student` objects. Students with the same `studentid` attributes are deemed equal.

7) Run the testing code provided in the Assignment Template. It should produce the following outputs (excluding the line numbers). Do not alter the testing code.

```
1. Invalid student id 1834633
2. Name: Dear Jane, ID: 21666666A
3. Name: Ka Yan 9896, ID: 21009896A
4. False
5. True
6. Invalid data type!
```

(b) A class `Registry` is used to wrap a list of 10 BSTs and its methods. The constructor is provided in the Assignment Template; it initializes a list of 10 empty BSTs. Write a method `add_student(name, studentid)` with the specification below. This method inserts a new student into the system. Another method `print_students()` is provided; it prints all students in the system (see lines 6-11 of the output below).

1) Take the variables `name` (student's name, `str`) and `studentid` (student's ID, `str`) as inputs. Validate their types.
2) Validate the input `studentid`. Print a message (without stopping the program) if invalid. See line 4 of the output below for the required format.
3) Identify the correct BST to use (out of the 10 BSTs) according to the last digit of the student ID.
4) Print a message (without stopping the program) if the student is already in the system. See line 5 of the output below for the required format.
5) Create a `Student` object with appropriate attribute values and insert it into the correct BST as the `data` field of a `BSTNode` (the class `BSTNode` is provided in the Assignment Template). The `key` field of the `BSTNode` is the `studentid`.
6) Check whether the insertion is successful by searching the BST to see if the same `Student` object (using "`is`" to check) can be retrieved. Print a message if successful. See lines 1-3 of the output for the required format.

```
1.  Register Chan Ka Yi 21123456A successful.
2.  Register So Fun Nei 21888888A successful.
3.  Register Ngor Mo Chin 21654321A successful.

4.  Invalid student id 81777778

5.  Chan Ka Yi 21123456A has already registered.

6.  Students with ID ended with 1:
7.  21654321A Name: Ngor Mo Chin, ID: 21654321A
8.  Students with ID ended with 6:
9.  21123456A Name: Chan Ka Yi, ID: 21123456A
10. Students with ID ended with 8:
11. 21888888A Name: So Fun Nei, ID: 21888888A
```

Question 4

This question is to build a data structure to store records of course enrollments. To this end, a singly linked list is used. The data structure is depicted in Figure 2.

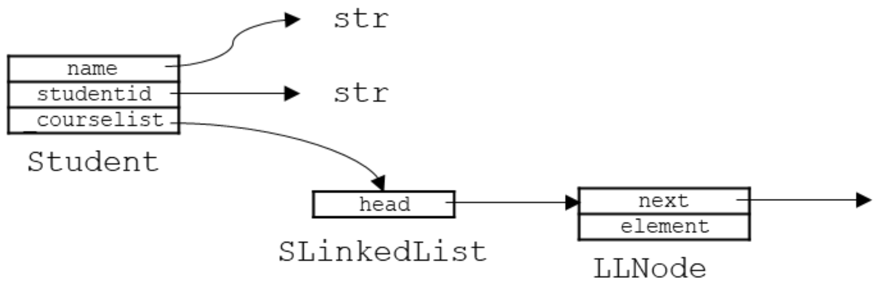

Figure 2.

(a) Define the `Student` class in Question 3(a) again, but with an additional attribute "`_courselist`" (with an underscore in front). Initialize this variable with an empty linked list implemented in the class `SLinkedList` (provided). The testing code should generate the following outputs (excluding the line numbers).

```
1.  SEHH2239
2.  SEHH2240
```

(b) A class `Registry2` is used to extend the class `Registry` in Question 3. It begins with the following definition to inherit the attributes and methods defined in `Registry`. The constructor calls the constructor of `Registry` to initialize a list of 10 empty BSTs.

```
class Registry2(Registry):
    def __init__(self):
        super().__init__()
```

Write a method `add_course(studentid, course)` with the following specification.

1) Take the variables `studentid` (student's ID, `str`) and `course` (course code, `str`) as inputs. Validate their types.
2) Validate the input `studentid`. Print a message (without stopping the program) if invalid. See line 8 of the outputs below for the required format.
3) Identify the correct BST to use (out of the 10 BSTs) according to the last digit of the student ID.
4) Retrieve the `Student` object by searching the BSTs with the key `studentid`.
5) Print a message (without stopping the program) if the student is not in the system. See line 9 of the outputs below for the required format.
6) Search the linked list to see if the student has already enrolled the course. If yes, print

a message See line 10 of the outputs below for the required format.

7) Insert the new course at the tail of the linked list.
8) Check whether the insertion is successful by searching the linked list to see if the node is found. Print a message if successful. See lines 4-7 of the outputs below for the required format.

```
1. Register Chan Ka Yi 21123456A successful.
2. Register So Fun Nei 21888888A successful.
3. Register Ngor Mo Chin 21654321A successful.

4. Enroll Chan Ka Yi 21123456A to SEHH2239 successful.
5. Enroll So Fun Nei 21888888A to SEHH2239 successful.
6. Enroll Ngor Mo Chin 21654321A to SEHH2240 successful.
7. Enroll Ngor Mo Chin 21654321A to SEHH2241 successful.

8. Invalid student id XDJCNDEG
9. 23000000A not a registered student.
10.Chan Ka Yi 21123456A has already enrolled SEHH2239.

11.Ngor Mo Chin 21654321A enrolled these courses:
12.SEHH2240
13.SEHH2241
```

(c) The data structure makes the retrieval of course enrollment records of a specific student efficient.
   (i) Comment on the efficiency of retrieving all students enrolled in a specific course. Give your answer in no more than 20 words.
   (ii) Do you have any suggestions to improve efficiency? Give your answer in no more than 30 words.

- END of Assignment 2 -