CSE4IP&CSE5CES(BU-1)
INTRODUCTION TO PROGRAMMING
PROGRAMMING FOR ENGINEERS AND SCIENTISTS
Semester 1, 2021
Assignment 2

Assessment: This assignment 2 is worth 25 % of the final mark for this subject.

Due Date: Sunday 30 May 2021 at 11.59 PM

Delays caused by computer downtime cannot be accepted as a valid reason for a late submission without penalty. Students must plan their work to allow for both scheduled and unscheduled downtime.

Penalties are applied to late assignments (accepted up to 5 working days after the due date only). See the university policy for details.

Individual Assignment: This is an individual assignment. You are not permitted to work as a group when writing this assignment.

Copying, Plagiarism: Plagiarism is the submission of somebody else's work in a manner that gives the impression that the work is your own. The Department of Computer Science and Information Technology treats academic misconduct seriously. When it is detected, penalties are strictly imposed. Refer to the unit guide for further information and strategies you can use to avoid a charge of academic misconduct. All submissions will be electronically checked for plagiarism.

Objectives: The general aims of this assignment are:

- To practise using concepts and techniques covered in the lectures and related labs.
- To apply those concepts and techniques for practical problem solving, i.e. to design and implement programming solutions

How to submit your assignment

1. The answer for Question 1 must be put in text file **A2Q1.py**. In this file, you write at the top of the answer: (i) Your student ID, (ii) Your first name, and (iii) Your last name, in that order. Do this for all other questions.
2. Put all your answers in a zip file, with the .zip extension. The name of the zip file should be CSE4IP-20001111 if your student Id is 20001111, for example.
3. Upload the zip file using Assignment 2 Submission set up on the subject's LMS.

Question 1– Translating Text Messages [10 marks]

Instant messaging (IM) and texting on portable devices has resulted in a set of common abbreviations useful for brief messages. However, some individuals may not understand these abbreviations.

Write a program that reads a one-line text message containing common abbreviations and translates the message into English using a set of translations stored in a file. For example, if the user enters the text message:

y r u l8

the program should print **why**

are you late

As a simplification, you can assume that are no punctuation marks.

Proceed as follows:

a. Build a dictionary with abbreviations as keys and associated texts as values.

Read the provided text file **abbreviations.txt**, each line of which contains an abbreviation and the associated text it.

(You should display the list **lines** to see what a string of this list looks like.)

Complete the code to build a dictionary with abbreviations as keys and the associated texts as value.

b. Translate a message.

Split the message into "words". If a "word" is in the dictionary, replace it by the associated text. Otherwise simply copy the "word" to the translation.

Question 2– Merge Big Files [10 marks]

Suppose we have two very big text files, so big that we cannot read them into the computer's memory. The files are already sorted.

Now, we want to merge them. How can we do this?

We can use the following approach. Imagine that the names are on two piles of cards.

- We look at the cards on top of the two piles.
- We then pick out the smaller one to add to the merged pile. If the two top cards have the same name, we add one card to the merged pile and discard the other one.

- We repeat this until one pile is exhausted. We simply add all the cards of the other pile, if any, to the merged pile.

Write a program to implement the above approach. For demonstration purpose, you can take **names set1.txt** and **names set2.txt** as the two input files.
Call the output file **names merged big.txt**.

> **Note:** The merge problem we consider here is part of the algorithm known as *external merge sort*, which is used to sort very big files, without reading the whole files into computer memory.

Question 3 – Pattern [10 marks]

Write a program in python to print the following pattern. You need to enter the number of rows from a user using **input()** function.

```
  1
  2    1
  4    2    1
  8    4    2    1
 16    8    4    2    1
 32   16    8    4    2    1
 64   32   16    8    4    2    1
128   64   32   16    8    4    2    1
```

Question 4– Date and Calendar [30 marks]

a. Define a function called **isLeapYear** that takes a year as an integer and returns **True** if it is a leap year and **False** otherwise. A year, between 1800 and 20000, is a leap year if
   - It is divisible by 4 but not by 100 *or*
   - It is divisible by 400

   For this function, you can assume that the year is between 1800 and 20000. You do not need to check it.

b. Define a function called **daysInMonth** that takes *a month of a year* and returns the *number of days* for that month. The function takes an integer for the month (1 for January, 2 for February, etc.) and a year (an integer between 1800 and 20000).

For this function, you can assume that the month is between 1 and 12 and the year is between 1800 and 20000. You do not need to check them.

c. Define a function called **isValidDate** that takes three integers for a day, a month and a year (in that order) and returns **True** if the thee integers represent a valid date and **False** otherwise.

   Among other things, this function must check that the day is between 1 and 31, the month is between 1 and 12, and the year is between 1800 and 20000. The function must return **False** if any of these conditions is not satisfied.

d. Let day, month and year be the day, the month and the year of a date. The day of the week this date falls in can be determined by the following method of calculation (given by the famous mathematician Carl Friedrich Gauss):

```
x = year - (14 - month)/12
y = x + x/4 - x/100 + x/400
z = month + 12 * ((14 - month) /12) -2
dow = (day + y + (31 * z)/12) % 7
```

   where the division is *integer division* (in which the fractional part is discarded), and **dow** represents the day of the week, with *Sunday being 0*, *Monday being 1*, etc. Write a function called **dayOfWeek** that takes the day, month and year of a date and returns a number to represent the day of the week for that date, with *0 for Monday*, *1 for Tuesday* and so on (Note the slight difference from the outcome of Gauss's algorithm). If the numbers do not represent a valid date the function should return **None**.

e. Write a function called **printCalendar** that takes two integers to represent a particular month in a particular year and prints out the calendar for that month. For this function, you can assume that the month is between 1 and 12 and the year is between 1800 and 20000.

f. Add statements to your program to ask the user for a birth date of a person (day, month and year). If the birth date is not valid, print the message

The given date is invalid.

Otherwise, display

- On which day of the week the person was born, e.g.

The person was born on a Monday

And

- The calendar of the *month of the birthday* in *2020*. Make sure that the columns (of the days of the week and the days) line up nicely.

Q: Should I submit more than one program for this question?

A: No, please put all the functions and the statements required for the last task in a program.

Q: What about test cases?

A: You are not required to do this. But it may be a good idea to include the testing code. But make sure you comment the code for testing out.

Question 5 – Reading Bus Route Data [40 marks]

In this question, you will work with a text file that contains information about the bus route that goes from the Bundoora campus of La Trobe University to the City of Melbourne.

The data (which is route 250) is stored in the text file **BusRoute250.txt**. It lists all the bus stops on the route, 67 of them actually. Details about a stop is stored on one line. The first 5 lines and the last 5 lines are shown below:

```
1/La Trobe Uni Terminus/Science Dr/Bundoora
2/La Trobe Uni Thomas Cherry Building/Science Dr/Bundoora
3/Kingsbury Dr/Waterdale Rd/Bundoora
4/Crissane Rd/Waterdale Rd/Heidelberg West
5/Percy St/Waterdale Rd/Heidelberg West ...
63/Exhibition St/Lonsdale St/Melbourne City
64/Melbourne Central/Lonsdale St/Melbourne City
65/Elizabeth St/Lonsdale St/Melbourne City
66/Little Bourke St/Queen St/Melbourne City
67/Little Collins St/Queen St/Melbourne City
```

Each line has four fields, separated by slash characters:

- The first is the stop number (an integer), counting from the La Trobe University end.
- The second is the stop's name. You may be interested in how bus stops are named. In general, the name of a bus stop can be
  - Either the name of a nearby street that cuts across the route (which will be referred to as the "cross-street"), e.g. Kingsbury Dr
  - Or the name of a place of interest near the stop (which will be referred to as the "cross-site"), e.g. La Trobe Uni Thomas Cherry Building
- The third is the street along the route on which the stop lies, e.g. Science Dr for stop number 1.
- The fourth field is the name of the suburb of the stop. It is enclosed between a pair of brackets, e.g. Bundoora.

A- Write a program to read the file and display the details about the bus stops. The details of a bus stop are displayed on four lines, showing the bus stop number, the name, the street, and the suburb. **[10 marks]**
For example:

```
1
La Trobe Uni Terminus
Science Dr
Bundoora ...
```

B- In this question, you write a number of functions to load the data into a list of bus stops and make various queries about the bus stops on the route. **[30 marks]**

Complete the program shown below:

```
def loadData():
       pass

def listAllBusStopNames(): pass

def listAllStreetsOnRoute(): pass

def listAllSuburbsOnRoute():
       pass

def searchByNumber(start: "int, value: a stop number", end: "int,
              value: a stop number >= start"):
       pass

def searchByName(nameKey): pass

def searchByStreet(streetKey): pass

def searchBySuburb(suburbKey): pass

def searchByAnyField(key):
       pass

#== Tests ==
# Add tests to test each of the functions
```

Please see descriptions of the functions on the next page.

loadData()

This function reads the data file **BusRoute250.txt**, creates the **BusStop** instances and put them in **busStopList**.

listAllBusStopNames()

This function lists all the names of the bus stops. They have to be listed in order starting from La Trobe University. Part of the output is shown below:

```
La Trobe Uni Terminus
La Trobe Uni Thomas Cherry Building
```

```
Kingsbury Dr
Crissane Rd
Percy St ...
Exhibition Building
Exhibition St
Melbourne Central
Elizabeth St
Little Bourke St
Little Collins St
```

listAllStreetsOnRoute()

This function lists all the streets along the route. The streets along the route correspond to the third field of the lines of the input data file. They have to be listed in order starting from La Trobe University. Part of the output is shown below:

```
Science Dr
Waterdale Rd
Dougharty Rd
Oriel Rd
Livingstone St ...
Bennett St
Holden St
Brunswick Rd
Rathdowne St
Lonsdale St Queen
St
```

listAllSuburbsOnRoute()

This function lists all the suburbs along the route. The streets along the route correspond to the the fourth field of the lines of the input data file. They have to be listed in order starting from La Trobe University. Part of the output is shown below:

```
Bundoora
Heidelberg West
Bellfield
Ivanhoe ...
Brunswick East
Carlton North
Carlton
Melbourne City
```

searchByNumber(start, end)

The function lists all the stops in this requested range. For example, if **start** is 5 and **end** is 10, the output should look like this:

```
Stop:5/Percy St/Waterdale Rd /Heidelberg West
Stop:6/Waterdale Rd/Dougharty Rd /Heidelberg West
```

```
Stop:7/Kolora Rd/Dougharty Rd /Heidelberg West
Stop:8/Ramu Pde/Oriel Rd /Heidelberg West
Stop:9/Outhwaite Rd/Oriel Rd /Heidelberg West Stop:10/Morobe
St/Oriel Rd /Heidelberg West
```

searchByName(nameKey)

This function lists all the bus stops whose names contain the search string **nameKey** as a substring. The matching ignores the difference between upper and lower cases. The matching bus stops are to be listed in the order starting from La Trobe University. A sample run is shown below for]cd nameKey being **de** :

```
Stop:8/Ramu Pde/Oriel Rd /Heidelberg West
Stop:12/Malahang Pde/Oriel Rd /Heidelberg West
Stop:37/Dennis Railway Station/Victoria Rd /Northcote
Stop:61/Carlton Gardens/Rathdowne St /Carlton
```

Note: If the search returns no bus stops, display message “There are no matches!”.

searchByStreet(streetKey)

Similar to the previous function, but the search is on the street along the route.

searchBySuburb(suburbKey)

Similar to the previous function, but the search is on the suburb along the route.

searchByAnyField(key)

This function lists all the bus stops whose name or street or suburb contains the search string **key**. Again, case differences are ignored, and the bus stops must be listed in the order starting from La Trobe University. A sample run is shown below for **key** being **gar**:

```
Stop:38/Westgarth St/Victoria Rd /Northcote
Stop:39/Simpson St/Westgarth St /Northcote
Stop:40/Northcote Park/Westgarth St /Northcote
Stop:41/Westgarth Railway Station/Westgarth St /Northcote
Stop:42/High St/Westgarth St /Northcote
Stop:61/Carlton Gardens/Rathdowne St /Carlton
```