NATIONAL JUNIOR COLLEGE
SENIOR HIGH 2 PRELIMINARY EXAMINATION
HIGHER 2

COMPUTING **9569/02**

Paper 2 (Lab-based) **28 August 2024**

3 hours

Additional Materials: Electronic version of `TASK1DATA.TXT` data file
Electronic version of `TASK2DATA.TXT` data file
Electronic version of `TASK3FILE.TXT` data file
Electronic version of `TASK4PEOPLE.CSV` data file
Electronic version of `TASK4PLAYERS.CSV` data file
Insert Quick Reference Guide

READ THESE INSTRUCTIONS FIRST

Answer **all** questions.

All tasks must be done in the computer laboratory. You are not allowed to bring in or take out any pieces of work or materials on paper or electronic media or in any other form.

Approved calculators are allowed.

Save each task as it is completed.

The use of built-in functions, where appropriate, is allowed for this paper unless stated otherwise.

Note that up to **6 marks out of 100** will be awarded for the use of common coding standards for programming style.

The number of marks is given in the brackets [ ] at the end of each question or part question. The total number of marks for this paper is 100.

This document consists of **17** printed pages.

[Turn Over

Instruction to candidates:

Your program code and output for each of Question 1 to 4 should be saved in a single `.ipynb` file using Jupyter Notebook. For example, your program code, output and answer for Question 1 should be saved as:

`TASK1_<your name>.ipynb`

Make sure that each of your `.ipynb` files shows the required output in Jupyter Notebook.

1 Name your Jupyter Notebook as:

`TASK1_<your name>.ipynb`

The task is to implement a Vigenère cypher encryption algorithm via the use of Caesar cypher.

A Caesar cypher encodes each letter with a different letter. A n-place Caesar cypher uses the ASCII value of each letter and adds the number n, where $1 \leq n \leq 26$ to it.

For example, if $n = 10$,

- The letter 'A' has the ASCII value 65. Adding 10 to the number will give the ASCII value 75. The character for 75 is 'K'.
- When an uppercase letter's code goes beyond 'Z' it returns to 'A'. For example, the character 'Z' will be encrypted as 'J'.
- When a lowercase letter's code goes beyond 'z' it returns to 'a'. For example, the character 'x' will be encrypted as 'h'.
- Spaces ' ' are replaced with the character '!'.

You will only need to convert letters and spaces. If the character is invalid, -1 is returned.

The following table shows the ASCII values of some of the characters.

Character	ASCII value
A	65
Z	90
a	97
z	122
”(space)	32
!	33

In ASCII the letters follow on numerically. For example, the letter ’A” is 65, ’B’ is 66, ’C’ is 67 etc.

For each of the sub-tasks, add a comment statement at the beginning of the code, using the hash symbol ’#’ to indicate the sub-task the program code belongs to, for example:

In [1]:
```
#Task 1.1

Program Code
```
```
Output:
```

Task 1.1

Write program code for a function `task1_1()` that:

- takes two parameters:
  - a character `c`, and
  - an integer n.
- encrypt the character `c` by adding the number n, where $1 \leq n \leq 26$, to its ASCII value.
- Spaces will be replaced with ’!’.
- returns the new encrypted character. If the character is invalid, -1 is returned. [5]

Task 1.2

The Vigenère cypher is a method of encrypting alphabetic text where each letter of the plaintext is encoded with a different Caesar cipher, whose increment is determined by the corresponding letter of another text, the keyword `key`.

For example, suppose that the plaintext to be encrypted is `'Attack#at dawn'` and the keyword is `'LEMON'`.

The keyword would be repeated until it matches the length of the plaintext.

```
Plaintext :  Attack#at dawn

Key       :  LEMONLEMONLEMO

Ciphertext:  Mygpqw#ni!pfjc
```

Because the 'L' is the twelveth letter in the alphabet, the first letter in the message ('A') is replaced by the letter which comes twelve places after it in the alphabet ('M'). In other words, 'A' is encrypted with 12-place Caesar cypher.

Because the 'E' is the fifth letter in the alphabet, the second letter in the message ('t') is replaced by the letter which comes five place after it in the alphabet ('y'). In other words, 't' is encrypted with 5-place Caesar cypher.

Because '#' is not an alphabet, it an invalid character and doesn't get encrypted.

And so on, until the complete message has been encrypted.

The text file `TASK1DATA.txt` contains a message that needs to be encrypted with the keyword `'Njc'` and then stored in a text file named `ENCRYPTEDMESSAGE.txt`

Write program code to:

- read the data from the text file `TASK1DATA.txt`
- encrypt each character with the Vigenère cypher and keyword `'Njc'`
- store the encrypted message in the text file `ENCRYPTEDMESSAGE.txt`

Keep the **invalid** characters as it is when appending to the encrypted message. [8]

Test your program with `TASK1DATA.txt`

Show the contents of `ENCRYPTEDMESSAGE.txt` after you have run the program. [1]

Save your Jupyter Notebook for Task 1.

2 Name your Jupyter Notebook as:

`TASK2_<your name>.ipynb`

A linked list is a collection of data elements, whose order is not given by their physical placement in memory. Instead, each element points to the next.

For each of the sub-tasks, add a comment statement at the beginning of the code, using the hash symbol '#' to indicate the sub-task the program code belongs to, for example:

In [1]:
```
#Task 2.1

Program Code
```
```
Output:
```

Task 2.1

Write the `LinkedList` class in Python. Use of a simple Python list is not sufficient. Include the following methods:

- `insert(data)` inserts the `data` at the beginning (head) of the list
- `delete(data)` attempts to delete `data` from the list; if the item was not present, return `None`
- `search(data)` returns a Boolean value: `True` if `data` is in the list, `False` if not in the list
- `count()` should return the number of elements in the list, or zero if empty
- `to_String()` should return a string containing a suitably formatted list with the elements separated by a comma and a space, with square brackets at either end, e.g. in the form: `[11, 2, 7, 4]`

[8]

Test `LinkedList` by using the integers in the file `TASK2DATA.TXT`. Use the `to_String()` method to print the resulting contents of the list. [3]

Task 2.2

In mathematics, a polynomial in x is an expression $p(x)$ consisting a single variable x and coefficients, that involves on the operations of addition, subtraction, multiplication and non-negative integer exponentiation of x.

For example, $x^2 - 4x + 7$, $8x^3 - 5x + 6$, x^5, 5 are polynomials in x.

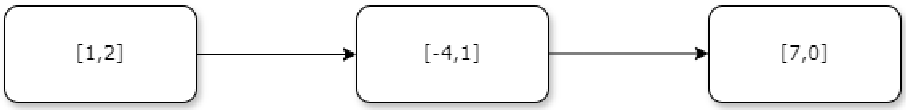
More generally, $p(x)$ is a polynomial in x if

$$p(x) = a_0x^0 + a_1x^1 + a_2x^2 + \cdots + a_nx^n,$$

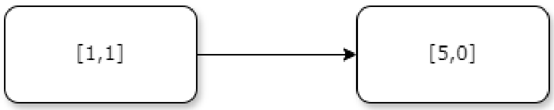
where $a_0, \cdots, a_n$ are the called *coefficients* of the term $x^0, \cdots, x^n$ respectively. The powers of x are called the *exponents*.

We can represent a polynomial in x as a linked list of nodes where each nodes contains 2 pieces of data, the coefficient and exponent.

Example 1, $x^2 - 4x + 7$ can be represented as:



Example 2, $x + 5$ can be represented as:



Write a Python subclass `Polynomial` using `LinkedList` as its superclass.

The `create` method in the `Polynomial` subclass takes in a list of data items, each data items contains a coefficient and an exponent. The method should ensure that the elements are stored in descending order of the exponent.

The `to_String` method in the `Polynomial` subclass should return a string that represents the polynomial in descending power of the exponent. Coefficients of the polynomials should be enclosed within parentheses characters `()`.

Example, the method should return the linked list representation of the polynomials:

- $x^2 + x^3 - 5$ as ''`(1)*x**3 +(1)*x**2 + (-5)*x**0`''
- $1 - x^4 + 2x$ as ''`(-1)*x**4 + (2)**x**1 + (1)*x**0`'' [6]

Test your program by creating `Polynomial` objects that represent the following polynomials:

- $p(x) = x^2 + x^3 - 5$,
- $q(x) = 1 - x^4 + 2x$,

Use the `to_String()` method to print the string representation of each of the polynomial. [2]

Task 2.3

Let $p\left(x\right)$ and $q\left(x\right)$ be polynomials such that

$$p\left(x\right) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n$$
$$q\left(x\right) = b_0 + b_1x + b_2x^2 + \cdots + b_mx^m$$

where m is not necessarily the same as n.

We define the addition of two polynomials $p\left(x\right) + q\left(x\right)$ to be

$$p\left(x\right)+q\left(x\right) = \begin{cases} \left(a_0+b_0\right)+\left(a_1+b_1\right)x+\cdots+\left(a_m+b_m\right)x^m+a_{m+1}x^{m+1}+\cdots+a_nx^n & \text{if } m \leq n, \\ \left(a_0+b_0\right)+\left(a_1+b_1\right)x+\cdots+\left(a_n+b_n\right)x^n+a_{n+1}x^{n+1}+\cdots+a_mx^m & \text{if } n < m \end{cases}$$

In other words, the resulting polynomial when two polynomials are added is a polynomial where the coefficients are the sum of the coefficients of $p\left(x\right)$ and $q\left(x\right)$ where the x term has the same exponent.

Amend your code for Task 2.2 to declare the `Polynomial` method `addPoly()` that takes a `Polynomial` object and returns another `Polynomial` object which represents the addition of the two polynomials. [7]

Test your `addPoly()` method on the Polynomial objects you created in Task 2.2 and use the `to_String()` method to print the string representation of the polynomial. [1]

Save your Jupyter Notebook for Task 2.

3 Name your Jupyter Notebook as:

`TASK3_<your name>.ipynb`

The task is to:

- read a file containing a list of numbers
- store them into a list
- sort the list using a merge sort
- choose n random elements from the list
- write the list of random elements to a file
- compute the average of these n random elements
- find the number of elements in the list that is less than average of these n random elements

For each of the sub-tasks, add a comment statement at the beginning of the code, using the hash symbol '#' to indicate the sub-task the program code belongs to, for example:

In [1]:
```
#Task 3.1

Program Code
```

```
Output:
```

Task 3.1

Write a function `task3_1(list_of_values)` that:

- takes a list of values, `list_of_values`
- sorts them into ascending order using merge sort
- returns the sorted list. [6]

Use the list `[56, 25, 4, 98, 0, 18, 4, 5, 7, 0]` to test your function.

For example, the condition

```
task3_1([56, 25, 4, 98, 0, 18, 4, 5, 7, 0]) == [0, 0, 4, 4, 5, 7, 18, 25, 56, 98]
```

should return `True`. [2]

Task 3.2

Write a function `task3_2(filename_in, filename_out, no_of_sample)` that :

- accepts three parameters:
  - `filename_in`, a string representing the input file name
  - `filename_out`, a string representing the output file name
  - `no_of_sample`, an integer representing the number of random elements to take from the list
- read the values from the input file `filename_in`
- store them into a list
- choose `no_of_sample` random elements from the list
- writes those values, one per line, to a file named `filename_out`
- calculates the arithmetic average of the `no_of_sample` random elements taken
- return the number of values found in file `filename_in` that is less than arithmetic average of the values found in `filename_out` [6]

Task 3.3

The collection of values of `no_of_sample` elements taken from the file `TASK3FILE.txt` is called *lower-sufficient* if the number of values in the file `TASK3FILE.txt` that is less than the average of the values in the collection, is less than an integer `alpha`. Else, the collection is called *lower-insufficient*.

Let `no_of_sample = 700` and `alpha = 1000`.

(a) Find a lower-insufficient collection and write it to the file `NOTLOWER.TXT`,

(b) Find a lower-sufficient collection and write it to the file `LOWER.TXT`. [4]

Save your Jupyter Notebook for Task 3.

4 Name your Jupyter Notebook as:

`TASK4_<your name>.ipynb`

A esports company has used a text file to store data collected about people who is organising a competition and the competitors. People who have an organising role at the competition are referred to as ‘staff’. Also, the players competes in teams of 5. The company decides to transfer this information into a database.

The database `ESPORTS` will have the following tables:

`PEOPLE`:

- `PersonID` - primary key, an auto-incremented integer
- `FullName` - the full name of the person, text
- `DateOfBirth` - the person’s date of birth, text
- `IsPlayer` - a Boolean using 0 for False and 1 for True, integer.
- `IsStaff` - a Boolean using 0 for False and 1 for True, integer.

`PLAYER`:

- `PersonID` - the person’s unique number
- `TeamName` - the player’s team name, text
- `CharacterName` - the person’s ingame name, text
- `EventName` - the person’s alias for the event, text
- `Score` - the person’s score for the competition, integer

A web page will then be used to summarise the data. Different information will be visible on the web page, depending on the type of person displayed.

Task 4.1

Write a Python program that uses SQL code to create the database `ESPORTS` with the two tables given. Define the primary and foreign keys for each table. [4]

Run your program to test the database has been set up correctly and save your database as `esports.db`. [1]

Task 4.2

The company wants to use the Python programming language and object-oriented programming to help publish the database content on a web page.

The class `Person` will store the following data:

- `full_name` - stored as a string
- `date_of_birth` - initialised with a string with the format YYYY-MM-DD

The class has three methods defined on it:

- `is_player()` - returns a string `'Maybe'`.
- `is_staff()` - returns a string `'Maybe'`.
- `event_name()` - returns a string which creates an identifier to be used as a event name, which should be constructed as follows:
  - the person's full name with all spaces and punctuation removed
  - followed by the two-digit month of their birth
  - then the two-digit day of their birth.

  For example, "`Bira Dawn`", born on the 02 october1985("`1985-10-02`"), would have theevent name `"BiraDawn1002"`

Save your program code as

`TASK4_2_<your name>.py` [4]

The `Player` class inherits from `Person`, such that:

- the class has three additional attributes
  - the player's character name `char_name`, and
  - the player's team name `team_name`,
  - the player's score `score`
- `event_name()`method, which returns a string which creates an identifier to be used as a event name, which should be constructed as follows:
  - the player's character name,
  - followed by a space character ' ' and the team name `'<TEAM_NAME>'`

  For example, `'Rogerbrown'`, from the team `'Echo'` would have the event name `"Rogerbrown <Echo>"`
- `is_player()` always returns True.

The `Staff` class inherits from `Person`, such that:

- `event_name()` should be the person's event name followed by `'Staff'`
- `is_staff()` always returns True.

Add your program code to

`TASK4_2_<your name>.py` [4]

Task 4.3

The text file, `TASK4PEOPLE.CSV`, contains data items for a number of people. Each data item is separated by a comma, with each person‘s data on a newline as follows:

- full name
- date of birth in the form YYYY-MM-DD
- a string indicating whether the person is `"Staff"`, `"Player"` or `"Person"`.

Another text file, `TASK4PLAYERS.CSV`, contains additional data items for the people who are also players. Each data item is separated by a comma, with each person‘s data on a newline as follows:

- full name
- date of birth in the form YYYY-MM-DD
- team name
- character name
- competition score

Write program code to read in the information from the text files, `TASK4PEOPLE.CSV` and `TASK4PLAYERS.CSV` creating an instance of the appropriate class for each person (either `Player`, `Staff` or `Person`). Store all this instances in list named `Event_People`. [6]

Write program code to insert all information from the text files, `TASK4PEOPLE.CSV` and `TASK4PLAYERS.CSV` into the two tables in `esports.db` database. Run the program. Add your program code to

`TASK4_2_<your name>.py` [8]

Task 4.4

The aggregated information from the database are to be displayed in a web browser.

Write a Python program and the necessary files to create a web application that enables the list of teams and the team members to be displayed on separate web pages.

For each team, the team's web page should include:

- the team name
- a table consisting of three columns.
  - The first column contains the character name of the team members,
  - The second column contains the event name of the team members,
  - The third column contains the player's competition score.

The players should be displayed in descending order of their scores.

Save your program as

`Task4_3_<your name>.py`

with any additional files / sub-folders as needed in a folder named

`TASK4_3_<your name>` [5]

Run the web application and save the output of the program as

`TASK4_3_<your name>.html` [3]