NATIONAL JUNIOR COLLEGE
Mathematics Department

General Certificate of Education Advanced Level
Higher 2

COMPUTING

Paper 2 (Lab-based)

9569/2
19 May 2023
3 hours

Additional Materials: Electronic version of data `CHARACTERS.TXT` file
Electronic version of data `ITEMS.TXT` file
Electronic version data `CACHE.TXT` file
Electronic version data `INTERNET.TXT` file
Electronic version data `IGP.CSV` 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 3 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 **14** printed pages and 2 blank pages.

Instructions to candidates:

Copy the folder from the thumb drive to the PC's desktop and rename the folder on the desktop to `<your name>`.(For example, TanKengHan). All the resource files are found in the folder and you should work on the folder in the desktop.
Your program code and output for each of Task 1 to 3 should be saved in a single `.ipynb` file. For example, your program code and output for Task 1 should be saved as `Task_1_<your name>.ipynb`.
You should have a total of **three** `.ipynb` files to submit at the end of the paper.
At the end of the exam, copy the working folder on your desktop to the thumb drive.

Task 1

In a multi-player role playing game, the characters in the game possessed personal attributes like name, class, strength, dexterity, intelligence and experience. Each character also owned pieces of items like weapons and tools. The data are stored as a collection of files with the following format:

File	File Data

`CHARACTERS.TXT`	Saved game character data in the following format:
	
	`<NAME>, <CLASS>, <STR>, <DEX>, <INT>, <XP>`
	`<ITEM 1 ID>, <ITEM 2 ID>, …, <ITEM N ID>`
	
	the data for each game character is saved in 2 lines of this text file.

`ITEMS.TXT`	Saved item data in the following format: `<ITEM ID>, <ITEM NAME>` the data for each item is saved in 1 line of this text file.

However, due to a system glitch, all the data in the files are stored in their encoded hexadecimal format, including formatting data i.e., commas(,) and end line characters(`\n`).

For example, a string: "`ABC,1,15\n`" would be stored as: "`4142432C312C31350a`".

With the conversion performed as follows:

CHARACTER	A	B	C	,	1	,	1	5	\n
ASCII (DEC)	65	66	67	44	49	44	49	53	10
ASCII (HEX)	41	42	43	2c	31	2c	31	35	0a

The files contain only ascii characters and are encoded in utf-8.

For each sub-task, 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

Implement a Python function `hex2Str(hex_str:str)` to return a string containing the decoded character data, `hex_str` is a string containing hexadecimal digits.

Example:

`hex2Str("4142432C312C31350a")` should return `"ABC,1,15\n"`.

You are allowed to use Python built-in functions in this task. [4]

Task 1.2

Write Python code to read, decode and store the contents of the file `ITEMS.TXT` in appropriate data structure/s such that they can be access efficiently later. Make use of the `hex2Str()` function to decode the contents of the file.

Print the contents of the data structure/s. [5]

Task 1.3

Write Python code to read, decode and store the contents of the file `CHARACTERS.TXT` in appropriate data structure/s such that they can be access efficiently later. Make use of the `hex2Str()` function to decode the contents of the file.

Print the contents of the data structure/s. [9]

Task 1.4

The contents of the two files `CHARACTERS.TXT` and `ITEMS.TXT` are to be migrated into a MongoDB database as a single collection.

Making use of the data structures that you have created in Task 1.2 and Task 1.3, write Python code to migrate the data into a collection named `Characters` and a database named `Overwatch`. The schema of the `Characters` collection is as follows:

```
{
 "name": "Aragon",
 "class": "Ranger",
 "str": "20",
 "dex": "25",
 "int": "20",
 "xp": "76525",
 "items": [{"id": "15", "name": "+4 SWORD"},
           {"id": "5", "name": "+2 BOW"} ]
}
```

[3]

Task 1.5

Use a pymongo query to find the names of all game characters that possessed an item named "+4 SWORD".

Write Python code to perform the query and print only the names. [3]

Task 1.6

After the data are migrated to MongoDB, the new version of the game supports unicode. A player can now include an alias name to his game character.

A player wishes to add an alias name "Ϯ∆Ψ" to his game character named "Aragon".

The unicode points for the alias characters are U+03EE, U+0394 and U+03A8 respectively.

Write Python code to perform the update.

[2]

Name and save your Jupyter Notebook as `Task1_<your name>.ipynb`.

Task 2

Caching is a technique used to speed up the response time of applications by storing frequently accessed data in memory, so that it can be quickly retrieved on subsequent requests without having to fetched the data from its source. A web browser typically uses cache data of frequently accessed web sites to improve its performance.

A hash table is to be implemented to map the key-value pairs of cached data. The URL of the web site is used as a key and the content or data from its web site is the value. When a user requests to fetch data from the web site, the web browser checks the hash table to see if the requested data is already cached. If the data is found in the hash table, it is returned immediately, else if the data is not in the cache, it is fetched from the web site and then added to the hash table using its URL as the key.

For each sub-task, 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

Implement a hash table class `HashTable` using Object-Oriented Programming (OOP) to store cached data for a web browser. The hash table contains the following methods :

- `constructor(n: integer)`
  - create an array of size `n` to store data in the hash table.
- `hasher(key: string): integer`
  - maps a key to an address in the hash table array using the following algorithm:

```
sum ← 0
FOR i = 0 to LEN(key)-1 DO
      sum += (ASCII value of key[i]) * (i+1)
ENDFOR
RETURN sum MOD n // n is the size of the array
```

- `insert(key:string, value:object): boolean`
  - the data to be inserted into the hash table consists of a key, value pair implemented as a Python tuple data type. The key will be used to determine the address or index in the hash table array to store the key, value pair tuple.
  - a linear probling algorithm will be used to resolve collisions.
  - returns `True`, if insertion is successful and `False` otherwise.
- `find (key: string ): object`
  - returns the object stored in the hash table using the key given.
  - returns `None` if the object is not found.
- `__repr__(): str`
  - returns the string representation of the hash table as follows:

    `[0:(key,value),1:(key,value), …5:(key,value) … ]`

    where 0,1,.. are the indexes of the array.

[17]

Task 2.2

Write Python code to test the implementation of the hash table by:

- create a hash table of size 3.
- add the following items into the hash table:
  - `("www.co", "web")`
  - `("sch.org", "school")`
  - `("ai.net "sky net")`
- Print the hash table.
- Print the results for finding "www.co" and "ai.net" in the hash table.

[2]

Task 2.3

A new hash table class named `OrderedHashTable` is to be implemented by inheriting from the `HashTable` class implemented in Task 2.1. This new class has an additional method, `keys()`, that will return a list of keys according to the order in which the data are inserted into the hash table. For example if the following code is executed:

```
hashT = OrderedHashTable(10)
hashT.insert("www.hi.cc", "Hello World")
hashT.insert("ppp.me", "test site")
hashT.insert("njc.sch", "school")
```

`hashT.keys()` will return `["www.hi.cc","ppp.me","njc.sch"]`.

In order to maintain the order of insertion, a logical linked list (NOT Python List) is to be implemented in the `OrderedHashTable` class by adding an attribute `head` which is an integer value representing the first item on the linked list and correspond to the index in the hash table array where the data is stored. If there are no insertions on the hash table, this value should be set to `-1`. The order of the keys inserted can then be retrieved by traversing this logical linked list. [10]

Task 2.4

Write the Python code to test the `keys()` method of the class `OrderedHashTable` using the example shown in Task 2.3. [1]

Task 2.5

To simulate a web browser's functionalities, perform the following tasks:

(a) Implement the procedure `load_cache(ht:OrderedHashTable)` that reads the contents in the file `CACHE.TXT` and insert them into the hash table implemented in Task 2.3.

(b) Test the `load_cache()` procedure by creating a hash table of size 6, load the contents of `CACHE.TXT` into the hash table and print the hash table.

(c) Implement the function,

`browse(url:string, cache:OrderedHashTable): RETURNS string`

that will search if the url is already stored in the cache, if found the function returns the content of the url. If not found, the file, `INTERNET.TXT` will be searched. If the url and its content is found in the file, the url and its content will be inserted into the cache and the function returns the content. If the url is not found in the file, the function returns "404 NOT FOUND".

(d) Test the `browse()` function by returning the contents for the url, "farm.net".

(e) Implement the function, `history(cache: OrderedHashTable): list` that returns a list of urls of web sites accessed on the Intrenet, starting with the last url accessed.

(f) Implement the procedure `clear_cache(cache: OrderedHashTable): OrderedHashTable` that will remove all the contents from the hash table. [10]

Task 3

The **incomplete** algorithm below performs an inplace merge on an array `A` (indexing is 0-based), of `n` elements. The input array `A` has elements from index `0` to `mid`, that is already sorted in ascending order and elements from index `mid+1` to `n-1` also sorted in ascending order. The algorithm will perform an inplace merge of the 2 sorted sub-array in `A` such that `A` will be sorted in ascending order from index `0` to `n-1`.

```
PROCEDURE inplace_merge(A:Array[0:n-1], start:integer, mid:integer,
end:integer)

 left ← start
 right ← mid + 1

 WHILE left < right AND right <= end DO

  IF A[left] <= A[right] THEN
   left ← left + 1
  ELSE
   index ← right
   tmp ← A[right]
   WHILE index > left DO
    A[index] ← A[index-1]
    index ← index -1
   END WHILE

   // Missing code

  END IF

 END WHILE

END PROCEDURE
```

For example,

```
A ← [2,4,6,8,1,3,5]

// subarray A[0:3] is sorted, subarray A[4:6] is sorted where A[i:j]

// denotes subarray from index i to j

CALL inplace_merge(A,0,3,6) // start is 0, mid is 3, end is 6
```

will result in a sorted array A, `[1,2,3,4,5,6,8]`.

Task 3.1

Implement the complete inplace merge algorithm show above in Python code. [3]

Task 3.2

Write test cases to test the correctness of the `inplace_merge()` function. [2]

Task 3.3

A recursive inplace merge sort can be implemented using the `inplace_merge` implementation in Task 3.1, as follows:

```
PROCEDURE merge_sort_helper(A, start, end)

 IF start >= end THEN
  RETURN
 END IF


// missing code

END PROCEDURE

PROCEDURE merge_sort(A: Array[0:n-1])

 start ← 0
 end ← n-1
 CALL merge_sort_helper(A, start, end)

END PRODECURE
```

Implement the `merge_sort()` and `merge_sort_helper()` in Python code. [4]

Task 3.4

Write test cases to test the correctness of the `merge_sort()` function. [2]

Name and save your Jupyter Notebook as `Task3_<your name>.ipynb`.

Task 4

When you apply for local universities in Singapore (NUS/NTU/SMU/SUTD/SIT), you will need to compute a score based on your A level results known as the University Admission Score (UAS).

You will need to provide the following inputs to compute your UAS.

- Grade for H1 General Paper(GP)
- Grade for H1 Project Work(PW)
- Grade for 3 H2 Subjects
- Grade for 1 H1 Subject (If you take 4 H2 subjects, the lowest H2 subject grade will be used as a H1 grade)

The following table shows the corresponding points for the grades you obtained for H1 and H2 Subjects in the Cambridge A level examination.

Grade	H2 Equivalent	H1 Equivalent
A	20	10
B	17.5	8.75
C	15	7.5
D	12.5	6.25
E	10	5
S	5	2.5
U	0	0

A perfect score of 90 points is obtained when you obtained A for H1 GP, H1 PW and 4 H2 subjects or 3 H2 and 1 H1 subjects:

10(GP) + 10(PW) + 60(3 H2) + 10(1 H1) = 90

You can assume that all the A level candidates took a minimum of 3 H2 subjects and 1 H1 subject and the compulsory H1 GP and H1 PW subjects.

Task 4.1

Create a Web Application using Python/Flask code for the users to enter their A level grades from a web browser and display the computed UAS.

Save your program as `Task4_1_<your name>.py`

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

`Task4_<your name>.` [8]

Task 4.2

Instead of using a web browser to enter the A level grades. A terminal-based client program is written to use socket-based communication to send the user input A level grades to a server program to compute the UAS score.

Write Python code to implement an iterative server that computes and return UAS score to be displayed on the client-program. You may re-use the code in Task 4.1.

The Python code for the client program is given below:

```
gp_grade = input("GP Grade:")
pw_grade = input("PW Grade:")
H2_1_grade = input("1st H2 subject grade:")
H2_2_grade = input("2nd H2 subject grade:")
H2_3_grade = input("3rd H2 subject grade:")
H1_grade = input("H1 Grade/Lowest H2 subject grade:")

s = socket.socket()
s.connect(("127.0.0.1",9999))
s.sendall( f"{gp_grade},{pw_grade},{H2_1_grade},{H2_2_grade},
{H2_3_grade},{H1_grade}".encode() + b"\n")
UAS = s.recv(1024)
print(f"UAS:{UAS.decode()}")
s.close()
```

Save your program as `Task4_2.py`. [4]

Task 4.3

The file `IGP.CSV` contains a partial list of the universities' courses and their minimum UAS points (cut-off points). The cut-off points in the file do not include the GP and PW subjects.

Write Python code to load the contents of the file into a database table.

The database name should be named `uni.db` and the table name should be `UAS`.

Save your program as `Task4_3.py`

in the same folder that you have created in Task 4.1 [3]

Task 4.4

Modify your Python code in part Task 4.1 so that a list of the courses that meet the minimum cut-off points based on your computed UAS is displayed in a table form. You should use the data from the database table that you created in Task 4.3**.** Note that the cut-off points provided in the file `IGP.CSV` only use 3 H2 and 1 H1 subjects for computation.

An example of the web display is as follows:

UAS Score (GP,PW,3H2,1H1): 82.5

Available Courses:

NUS	Engineering
NUS	Architecture
NUS	Business Administration
NTU	Engineering
NTU	Computer Engineering
NTU	Aerospace
NTU	Physics
NTU	Sports Science
SMU	Computer Science
SMU	Economics

Save your program as `Task4_4_<your name>_<NRIC number>.py`

in the same folder that you have created in Task 4.1 [5]

END OF PAPER