

ANGLO-CHINESE JUNIOR COLLEGE
JC2 PRELIMINARY EXAMINATION

Higher 2

COMPUTING

9569/02

Paper 2 (Lab-based)

12 August 2022

3 hours

Additional Materials: Electronic version of `words.txt` data file
 Electronic version of `Task1_5.txt` data file
 Electronic version of `Task4.db` data file
 Electronic version of `books_data.txt` data file
 Electronic version of `copies_data.txt` 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 brackets [] at the end of each question or part question.
The total number of marks for this paper is 100.

This document consists of **9** printed pages and **1** blank page.



Anglo-Chinese Junior College

[Turn Over

Instruction to candidates:

Your program code and output for each of Tasks 1, 2 and 3 should be downloaded in a single .ipynb file. For example, your program code and output for Task 1 should be downloaded as TASK1_<your name>_<centre number>_<index number>.ipynb

- 1 Wordle is a popular word game where players are made to guess a five-letter word within six tries. The task is to recreate the game in Python.

Task 1.1

Write a function `get_word(filename)` that:

- accepts `filename`, a string representing the name of a file containing one or more five letter words;
- randomly chooses a word from the file and returns the word. [3]

Test the function with `words.txt`.

For example,

```
get_word('words.txt')
```

should return a word randomly chosen from `words.txt`. [1]

Task 1.2

Write a function `check_validity(guess)` to check if `guess` is valid.

A valid guess should:

- contain only lowercase letters;
- not contain numbers, punctuation, spaces or any other symbols;
- be exactly five letters in length.

The function would return `True` if `guess` is valid and `False` if not. [3]

Test the function fully with suitable test cases. [2]

Task 1.3

Write a function `check_guess(guess, word)` that:

- accepts two parameters:
 - `guess`: a string that represents what the player guessed;
 - `word`: a string that represents the answer;
- creates two lists:
 - the list `correct` is a list of indices of letters in `guess` that are in `word` and in the correct position;
 - the list `in_word` is a list of indices of letters in `guess` that are in `word` but not in the correct position;
- returns a list containing `correct` and `in_word` [6]

Test the function with the following test cases:

```
check_guess('maple', 'apple') should return [[2,3,4],[1]].
check_guess('poppy', 'apple') should return [[2],[0]].
check_guess('apple', 'apple') should return [[0,1,2,3,4],[ ]].
check_guess('apepe', 'apple') should return [[0,1,4],[3]].
```

[3]

Task 1.4

Write a function `display_result(correct,in_word)` to inform the player of the results of his guess.

The function:

- accepts two parameters, the lists `correct` and `in_word`;
 - outputs a string containing only the following characters: `^*_`, where
 - `^` represents a letter that is in the word and in the right position;
 - `*` represents a letter that is in the word but not in the right position;
 - `_` represents a letter that is not in the word;
- [4]

The output of the function should be displayed to the player.

Test the function with the following test cases:

```
display_result([2,3,4],[1]) should display '_*^^^'.
display_result([0,1,2,3,4],[ ]) should display '^^^^^'.
display_result([], [1]) should display '_*____'.
```

[3]

Task 1.5

The function `play_wordle()` for the game can be found in `Task1_5.txt`. The function `play_wordle()` makes use of the functions you created in **Tasks 1.1 to 1.4** to run the game.

Copy the `play_wordle()` function from `Task1_5.txt` into your Jupyter Notebook.

Run the program. [1]

Download your program code and output for Task 1 as

`TASK1_<your name>_<centre number>_<index number>.ipynb`

2 Binary search trees can be stored in a two-dimensional array, where each row of the array represents a node and contains the following information:

- the data (an integer) stored within that node;
- the left pointer, an integer which is the index of the left child of that node;
- the right pointer, an integer which is the index of the right child of that node.

Unused rows within the array are stored as a free list, a linked list where each node's left child is the next item in the list.

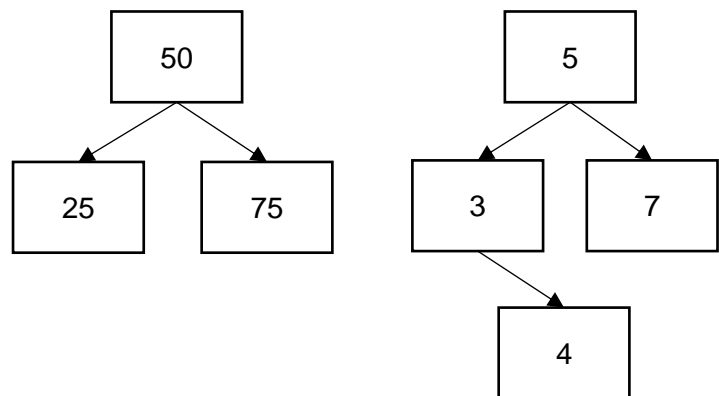
For instance, the diagram below shows how two trees are stored within an array.

Index	Data	Left	Right
0	50	2	6
1	5	8	3
2	25	None	None
3	7	None	None
4	4	None	None
5	None	9	None
6	75	None	None
7	None	5	None
8	3	None	4
9	None	None	None

Free list pointer: 7

Tree 1 pointer: 0

Tree 2 pointer: 1



Task 2.1

The array is initialised with n rows, where n is determined by the user when the array is created.

Upon initialisation, the data in each row is `None`, and the left pointer in each row points to the row immediately below it. The left pointer of the final row is `None`. The free list pointer points to the first row.

Write code to create an `Array` class as described above. There should be accessor and mutator methods for the free list pointer, as well as the data, left and right pointers for each row. [9]

Task 2.2

A BST (Binary Search Tree) is initialised using an existing `Array` object as an argument. It is initialised as an empty tree, with a pointer value of `None`.

Create a `BST` class.

[2]

Write code for the following methods:

- `add_node(data)` takes the first available node from the free list, inserts `data` as the node's data, and adds the node to the appropriate location in the BST. If the free list is empty, an appropriate message should be output. It may be assumed that the new node does not have the same data as any of the existing nodes in the BST. [10]
- `search_tree(data)` searches the BST for a node with `data`, and returns the index of that node within the `Array`. If the data does not exist within the BST, an appropriate message should be output. [6]
- `in_order_traversal()` returns the data in the nodes of the BST, following an in-order traversal. [5]

Your methods should demonstrate encapsulation of the `Array` object.

Download your program code and output for Task 2 as

`TASK2_<your name>_<centre number>_<index number>.ipynb`

- 3 In this task, you are **not** allowed to use the keyword `in` to determine whether a character exists in a string, or whether one string is a substring of another (e.g. `'G' in 'SINGAPORE'` or `'GAP' in 'SINGAPORE'`). You are also not allowed to check directly if two substrings longer than one character are equal (e.g. `'SINGAPORE'[0:3] == 'HELSINKI'[3:6]`).

The Boyer-Moore algorithm was designed to check if a string s is a substring of another string t (called the text).

Task 3.1

Write a function `dic(s)` that creates a dictionary whose keys are the characters of s , and the value of each key is the difference between the index of its last occurrence in s , from the largest index in s .

For example, `dic('anman')` creates a dictionary with the following keys and values:

Key	Value
'a'	1
'n'	0
'm'	2

[3]

Task 3.2

To determine whether s is a substring of t , we first compare the last letter of s with the character in t that has the corresponding index. The following shows two examples.

```

      *
t1: a n p a n m a n      *
s : a n m a n
      *
t2: a n p m n m a n      *
s : a n m a n
      *
```

If they match, we move on to the second last letter, and so on, until there is a mismatch.

```

      *
t1: a n p a n m a n      *
s : a n m a n
      *
t2: a n p m n m a n      *
s : a n m a n
      *
```

```

      *
t1: a n p a n m a n
s : a n m a n
      *
```

If we reach the first character of s and there is no mismatch, then s is a substring of t . If there is a mismatch (as illustrated in both examples above), then we shift s forward.

For t_1 , the mismatched character (`'p'`) does not appear in s at all. Therefore, we can shift s forward until the first character of s is after the mismatched character, and start checking again from the last character of s . In this case, we find that we can reach the first character of s without a mismatch, and therefore s is a substring of t .

```

      *
t1: a n p a n m a n
s :      a n m a n
      *
```

For t_2 , the mismatched character ('m') occurs in s . We therefore need to shift s so that the last occurrence of 'm' in ' s ' lines up with the 'm' in t_2 , and start checking again from the last character of s .

```

          *
t2: a n p m n m a n
s :   a n m a n
          *
```

In this case, there is a mismatch immediately, and the mismatched character ('m') occurs in s . We therefore need to shift s so that the last occurrence of 'm' in ' s ' lines up with this 'm' in t_2 .

```

          *
t2: a n p m n m a n
s :       a n m a n
          *
```

We start checking again from the last character of s .

There is an edge case in which the mismatch happens in which the mismatched character in t ('a') is a character which occurred to its right in s .

```

          * (mismatched character is the 'a')
t3: a n p a m a n m a n
s :       a n m a n
          *
```

(result of applying the shifting rule)

```

t3: a n p a m a n m a n
s : a n m a n
```

In this case, the shifting rule would shift s backwards, which is counterproductive. In this case, the best we can do is shift s one character forward and start checking again from the last character.

```

          *
t3: a n p a m a n m a n
s :       a n m a n
          *
```

The loop of checking and shifting continues until we shift s so that its last character is beyond the last character of t .

Write code to carry out this algorithm to check if s is a substring of t . [10]

Test your code on the following test cases:

```

s = 'anman'
t1 = 'aman'
t2 = 'anpanman'
t3 = 'anpanpan'
t4 = 'anpamanman'
```

[1]

Download your program code and output for Task 3 as

TASK3_<your name>_<centre number>_<index number>.ipynb

- 4 A library wants to store the details of its books in a relational database. The tables for the database have been created and stored in `Task4.db`.

There are two tables in the database:

- `books(bookID, title, price)`
- `copies(copyID, bookID)`

The library currently stores its information in two text files, `books_data.txt` and `copies_data.txt`. The library would like to reduce duplication of information by storing all the data into the relational database, `Task4.db`.

Each line in `books_data.txt` contains information regarding a book in the following format:

```
bookID,title,price
```

Each line in `copies_data.txt` contains information regarding a copy of a book in the library in the following format:

```
copyID,bookID,title,price
```

Task 4.1

Write a Python program to insert all information from the files into `Task4.db`.

Run the program.

[4]

Save your program code as

`TASK4_1_<your name>_<centre number>_<index number>.py`

Task 4.2

Write a Python program and the necessary files to create a web application. The web application offers the following menu options.

Insert a new book Display all books
--

Save your program code as

`TASK4_2_<your name>_<centre number>_<index number>.py`

with any additional files/subfolders as needed in a folder named

`TASK4_2_<your name>_<centre number>_<index number>`

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

`TASK4_2_<your name>_<centre number>_<index number>.html`

[3]

Task 4.3

The library would like to extend the web application to:

- receive `bookID`, `title`, `price`, and number of copies for a new book to be inserted into `Task4.db`;
- assign a `copyID` to each copy of the book;
- insert the data into the respective tables of `Task4.db`.

The `copyID` is a four-digit string starting with 0001. For example, if a book has three copies, they would have the `copyIDs`

- 0001
- 0002
- 0003

respectively.

The web page to receive the required information should be accessed from the corresponding menu option from **Task 4.1**. [6]

Test your program by inserting a book with the following details into `Task4.db`.

- `bookID`: 567420
- `title`: Plant farm
- `price`: 15.00
- number of copies: 4

[1]

Save your program code as

`TASK4_3_<your name>_<centre number>_<index number>.py`

With any additional files/subfolders as needed in a folder named

`TASK4_3_<your name>_<centre number>_<index number>`

Save your Python program as

`TASK4_3_<your name>_<centre number>_<index number>.py`

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

`TASK4_3_<your name>_<centre number>_<index number>`

Task 4.4

The library would like the web application to display information for all the books in a table.

Extend your web application to show the `bookID`, `title`, `price`, and the number of copies of that book in a table. The results should be shown in alphabetical order by book title.

The resulting web page should be accessed from the corresponding menu option from **Task 4.1**. [6]

Save your program code as

`TASK4_4_<your name>_<centre number>_<index number>.py`

with any additional files/subfolders as needed in a folder named

`TASK4_4_<your name>_<centre number>_<index number>`

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

`TASK4_4_<your name>_<centre number>_<index number>.html`

[2]