



YISHUN INNOVA JUNIOR COLLEGE
JC 2 PRELIMINARY EXAM
Higher 2

CANDIDATE
NAME

CG

INDEX NO

COMPUTING

Paper 2 (Lab-based)

9569/02

27 August 2020

3 hours

Additional Materials: Removable storage device with the following files:

- SONG.TXT
- PSEUDOCODE_TASK_3_3.TXT
- IN.JPG
- OUT.JPG

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.

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

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**.

Save a copy of all the files containing the program codes in the thumb drive provided.

Instruction to candidates:

Your program code and output for each of Task 1 to 4 should be downloaded in each 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

For each of the subtasks, 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:

In [2]:

#Task 2.2
Program Code

Output:

In [3]:

#Task 2.3
Program Code

Output:

- 1 In a Battleship game, a player fires missiles within a region measuring 6 metres by 6 metres, represented with full-stops (". "), to sink the ships. Each ship is represented by "XXX".

The region is represented on the screen by a rectangular grid. Each square metre of the region is represented by an x-coordinate and a y-coordinate. The top left square metre of the region display has $x = 0$ and $y = 0$.

Task 1.1

In one of the games, three ships were positioned in the region as given in the text file `GAME.TXT`. Write a program to read in the data from this file, store it in a suitable data structure and display on the screen as shown.

```
. . . . .
. . . . X .
X X X . X .
. . . . X .
. X X X . .
. . . . .
```

[6]

Task 1.2

During the game, the player cannot see the ships in the region.

```
. . . . .
. . . . .
. . . . .
. . . . .
. . . . .
. . . . .
```

Write program code to allow the player to fire 5 missiles targeting at specific locations, one at a time after observing each outcome. The player will input an x-coordinate and y-coordinate for each targeted location.

If the missile strikes any part of a ship, the damaged square metre will be represented with an "s", otherwise an "o" to represent it has missed. A sunken ship will be represented by "sss".

one missile which missed the ship

```
O . . . . .
. . . . .
. . . . .
. . . . .
. . . . .
. . . . .
```

another missile which struck the ship

```
O . . . . .
. . . . .
. S . . . .
. . . . .
. . . . .
. . . . .
```

The program code should also display the positions of all the ships at the end of the game.

```
O . . . . .
. . . . X .
S S S . X .
. . . . X .
. X X S . .
. . . . .
```

[7]

Download your program code and output for Task 1.1 and 1.2 as

Task1_<your name>_<centre number>_<index number>.ipynb

Task 1.3

Write server and client program for this asymmetric Battleship game where a server display the region and the player (the client) selects the target locations for his missiles. After firing each missile, the server returns an updated display for the region indicating a strike or a miss. Once the player fires his last missile, the server will display the positions of all the ships, together with the damages and misses, and both the client and server quit the game.

[12]

Download your server and client program code for Task 1.3 as

Task1_server_<your name>_<centre number>_<index number>.py

Task1_client_<your name>_<centre number>_<index number>.py

- 2 The file `SONG.TXT` contains the lyrics of the song, *Count on me Singapore*. The task is to read every word from the file, store it in a suitable data structure, sort the words and perform searches based on word and count.

Task 2.1

Write program code to:

- read the words from the file and store them in a suitable data structure,
- sort the words in alphabetical order using **quick sort**,
- write each word and their number of occurrence in a text file, `WORDCOUNT.TXT`, where the next word is on a new line. [12]

A sample of the `WORDCOUNT.TXT` for the first 5 lines is as follows:

```
a 5
achieve 12
air 1
all 1
and 9
```

Task 2.2

Write program code to:

- read the words and counts from the `WORDCOUNT.TXT` file
- allow the user to select the following options:
 1. Search for a word
 2. Search for word(s) based on count
 3. Quit program
- take in user input for the word or the count
- if found, display the word and its count, else display "Not Found"
- display appropriate error messages for invalid user input [6]

Task 2.3

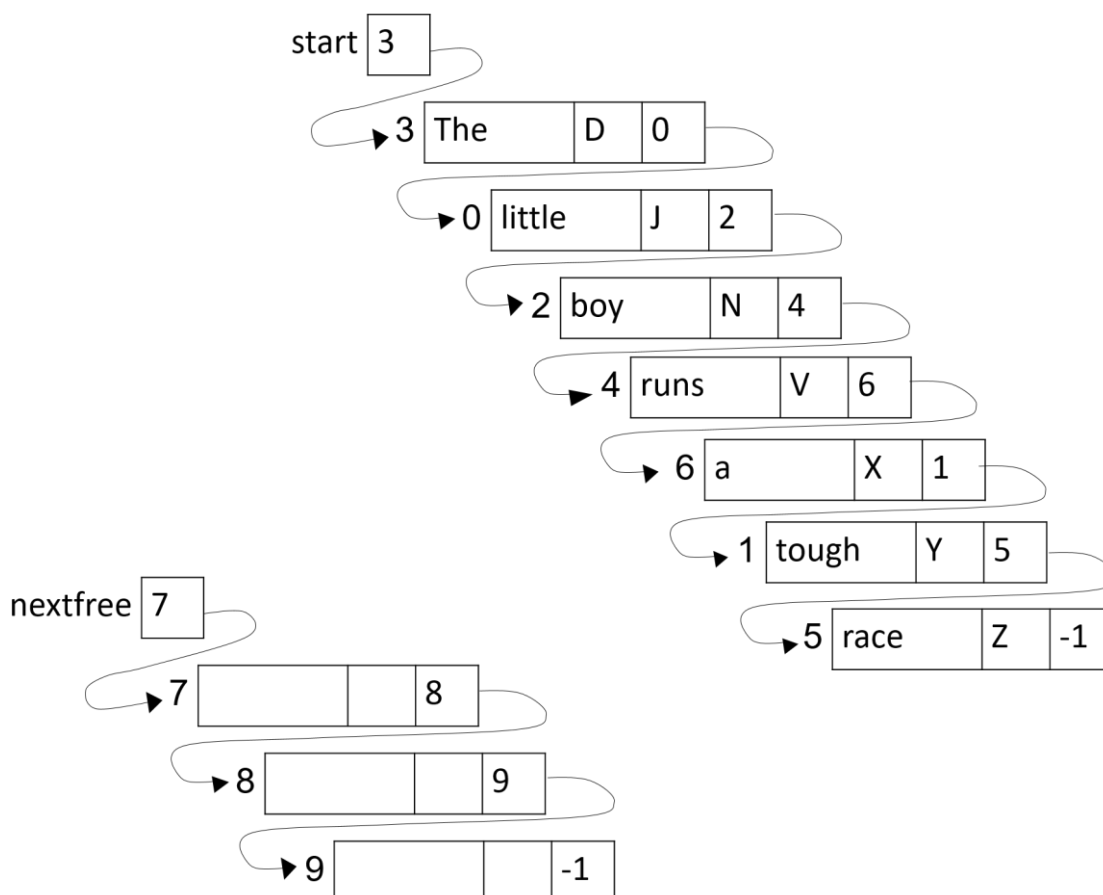
Design test data for your program written in **Task 2.2**, provide evidence of testing that includes:

- search for a word that is contained in the file
- search for a word that is not contained in the file
- search for word(s) with a count that is contained in the file
- search for word(s) with a count that is not contained in the file [2]

Download your program code and output for Task 2 as

Task2_<your name>_<centre number>_<index number>.ipynb

- 3 The task is to store words in nodes that is contained within a linked list data structure. A node contains a word, the word category and a next pointer. The pointers link the word items in proper grammatical order based on their word category ('N': noun, 'V': verb, 'D': determiner, 'J': adjective). The unused nodes are linked as shown below. The first unused node is the position where the new word item is to be stored.

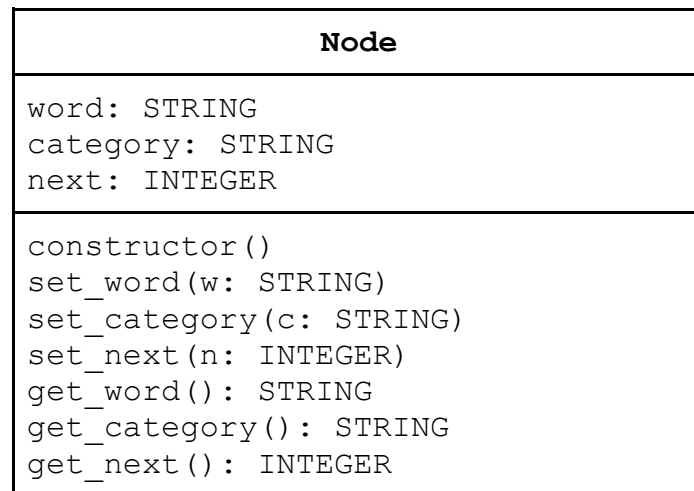


The diagram shows the linked list with:

- the words and their respective category inserted in the following order:
 - 'little', 'J'
 - 'tough', 'J'
 - 'boy', 'N'
 - 'The', 'D'
 - 'runs', 'V'
 - 'race', 'N'
 - 'a', 'D'
- the unused nodes linked together

Each node is implemented as an instance of the class Node.

The class Node has the following UML class diagram:



The LinkedList class is implemented as follows:

Class: LinkedList		
Attributes		
Identifier	Data Type	Description
sentence	ARRAY[0:9] of Node	The linked list data structure that contains 10 nodes.
start	INTEGER	Index for the start position of the linked list.
nextfree	INTEGER	Index for the next unused node.
Methods		
Identifier		Description
__init__	PROCEDURE	Sets all the node data values to 'empty string'. Set pointers to indicate all nodes are unused and linked. Initialise values for <code>start</code> and <code>nextfree</code> .
isempty	FUNCTION RETURNS BOOLEAN	Tests for empty linked list.
isfull	FUNCTION RETURNS BOOLEAN	Tests for no unused nodes.
display	PROCEDURE	Displays the contents of <code>sentence</code> in index order.
insert	PROCEDURE	Adds a new word and its category to the linked list.
traversal	PROCEDURE	Displays the simple sentence obtained from the linked list.

The index of the first available node is indicated by `nextfree`. The initial values of `start` and `nextfree` is -1 and 0 respectively .

Task 3.1

Write the program code to define the `Node` and `LinkedList` classes.

Do not attempt to write the methods `insert` and `traversal` at this stage. [10]

Task 3.2

Write program code to create a `LinkedList` object and run the `display` method to confirm the initial values of the pointers, word and category values when the linked list is empty. [2]

A simple sentence contains words from different category arranged in the manner as illustrated:

(‘N’: noun, ‘V’: verb, ‘D’: determiner, ‘J’: adjective)

Verb come between two nouns.

Determiner comes before a noun, adjective comes before a noun and after a determiner.

boy	runs	race
N	V	N

The	boy	runs	a	race
D	N	V	D	N

The	little	boy	runs	a	tough	race
D	J	N	V	D	J	N

In order to aid the process of inserting the words in their correct position in the linked list, the code letter ‘X’, ‘Y’ and ‘Z’ is used in place of the second set of ‘D’, ‘J’, ‘N’ respectively, so that the correct position can be determined by comparing the category when traversing down the linked list.

The	little	boy	runs	a	tough	race
D	J	N	V	X	Y	Z

The following pseudocode (available in PSEUDOCODE_TASK_3_3.TXT) can be used to add a word and its category to the linked list.

```
PROCEDURE insert(new_word, new_category)

    //check if Linked List is full
    IF nextfree = -1 THEN
        RETURN 'Linked List is Full'
    ENDIF

    //linked list is not full, safe to insert new word
    sentence[nextfree].word ← new_word
    sentence[nextfree].category ← new_category

    IF start = -1 THEN          //inserting into empty list
        start ← nextfree
        temp ← sentence[start].next
        sentence[start].next ← -1
        nextfree ← temp

    ELSE
        // traverse down the linked list to search for position to
        // insert
        current ← start //pointer of current node
        previous ← -1 //pointer of previous node
        inserted ← FALSE //flag to check for insertion

        WHILE current > -1 AND inserted = FALSE
            IF sentence[current].category > new_category THEN

                //position found, insert before current node
                IF current = start THEN
                    //check if current equals to start
                    start ← nextfree
                ELSE
                    sentence[previous].next ← nextfree
                ENDIF

                temp ← sentence[nextfree].next
                sentence[nextfree].next ← current
                nextfree ← temp
                inserted ← TRUE

            ELIF sentence[current].category < new_category THEN
                previous ← current
                current ← sentence[current].next

            ELSE THEN
                IF new_category = 'N' THEN
                    new_category ← 'Z'
```

```

ENDIF
IF new_category = 'D' THEN
    new_category ← 'X'
ENDIF
IF new_category = 'J' THEN
    new_category ← 'Y'
ENDIF

previous ← current
current ← sentence[current].next
sentence[nextfree].category ← new_category
ENDIF
ENDWHILE

IF inserted = False THEN
    sentence[previous].next ← nextfree
    temp ← sentence[nextfree].next
    sentence[nextfree].next ← -1
    nextfree ← temp
ENDIF

ENDIF
ENDPROCEDURE

```

Task 3.3

Write code to implement `insert` method from this pseudocode.

You may use the text file `PSEUDOCODE_TASK_3_3.TXT` as a basis for writing your code. [8]

Task 3.4:

Write code to insert the following into the linked list created in **Task 3.2** and display its contents:

- o 'little', 'J'
- o 'tough', 'J'
- o 'boy', 'N'
- o 'The', 'D'
- o 'runs', 'V'
- o 'race', 'N'
- o 'a', 'D'

[2]

Task 3.5:

Write code to implement traversal method which displays the simple sentence obtained from traversing the linked list and run it.

The expected output should look like this:

The little boy runs a tough race.

[8]

Download your program code and output for Task 3 as

`Task3_<your name>_<centre number>_<index number>.ipynb`

- 4 *SafeEnter* is a digital check-in system that tracks the people visiting the public places, to prevent and control the transmission of COVID-19 through contact tracing.



Task 4.1

Create a HTML file called `index.html` to display the Check-In form for people to input their particulars and other details. (Use the picture `IN.JPG` provided.) [5]

The image shows a web form titled 'SafeEnter' with a red logo above the text. The form is enclosed in a light blue border and contains five input fields, each with a label to its left: 'Postal Code :', 'Date (yyyymmdd) :', 'Check-in Time (hh:mm) :', 'NRIC :', and 'Handphone No. :'. Each input field is a simple rectangular box. At the bottom left of the form, there is a grey button with the text 'Submit' in black.

Check-In Form


Task 4.2

Write program code, `server.py`, for the back-end server to display the Check-In form on the clients' browser when they scan a QR-code which links to the *SafeEnter* website.

The server script should include a route `/checkin` to receive the inputs from the client, the program code should:

- prevent the user from accessing the `/checkin` route directly
- receive all the inputs in the Check-In form
- reject empty or null inputs

- append the new entry into the `event` table in the **given** database `covid.db`
- reply by sending a `checkout.html` page back to the client's browser, displaying the following Check-Out form, which is to be used when leaving the venue (Use the picture `OUT.JPG` provided.) [12]



Postal Code :

Date (yyyymmdd) :

Check-out Time (hh:mm) :

NRIC :

Handphone No. :

Check-Out Form

Task 4.3

Modify the program code, `server.py`, and include an additional route `‘/checkout’` to receive the inputs from the client's Check-Out form when they leave the venue. The program code should:

- allow the user from accessing the `‘/checkout’` route directly
- receive all the inputs in the Check-Out form
- reject empty or null inputs
- update the corresponding entry in the `event` table in the **given** database `covid.db` [8]

Download the files for your program codes for Task 4 as

```
Task4_<your name>_<centre number>_<index number>_index.html
Task4_<your name>_<centre number>_<index number>_checkout.html
Task4_<your name>_<centre number>_<index number>_server.py
Task4_<your name>_<centre number>_<index number>_covid.db
```