UNIVERSITI MALAYA
UNIVERSITI MALAYA

PEPERIKSAAN IJAZAH SARJANA MUDA SAINS KOMPUTER
EXAMINATION FOR THE DEGREE OF BACHELOR OF COMPUTER SCIENCE

SESI AKADEMIK 2023/2024 : SEMESTER II
ACADEMIC SESSION 2023/2024 : SEMESTER II

WIA1002: Struktur Data
Data Structure

Jun/Julai 2024 MASA: 3 jam 30 minit
June/July 2024 TIME: 3 hours and 30 minutes

ARAHAN KEPADA CALON:
INSTRUCTIONS TO CANDIDATES:

(1) Calon dikehendaki menjawab **SEMUA** soalan.
*Candidates should answer **ALL** questions.*

(2) Kertas soalan ini mengandungi **2 BAHAGIAN**: A dan B. (50 markah)
*This question paper contains **2 SECTIONS**: A and B. (50 marks)*

Bahagian A	**Soalan 1 – 40 (20 markah)**
Section A	***Question 1 – 40 (20 marks)***
Bahagian B	**Soalan 41 – 43 (30 markah)**
Section B	***Question 41 – 43 (30 marks)***

(3) Semua jawapan **mesti dihitamkan** di kertas jawapan OCR yang disediakan dengan menggunakan **pensil 2B SAHAJA**.
*All answers **must be shaded** on the provided OCR form using **2B pencil ONLY**.*

(4) Peringatan: Kertas soalan ini **TIDAK DIBENARKAN** dibawa keluar dari Dewan Peperiksaan.
*Reminder: This question paper is **NOT ALLOWED** to be taken out from the Examination Hall.*

(Kertas soalan ini mengandungi 43 soalan dalam 36 halaman yang dicetak)
(This question paper consists of 43 questions on 36 printed pages)

BAHAGIAN B
SECTION B

Jawab Soalan 41 sehingga Soalan 43 menggunakan Netbeans dan hantar semua jawapan anda berdasarkan arahan yang diberikan oleh pengawas peperiksaan.

Answer Question 41 to Question 43 using Netbeans and submit all answers according to the instructions given by the invigilators.

41.a) Anda diberikan kelas **Node** dan kelas **DoublyLinkedList** seperti berikut:
*You are given the following **Node** class and **DoublyLinkedList** class:*

Kelas **Node**
***Node** class*

```
class Node {
    int data;
    Node prev;
    Node next;

    Node(int data) {
        this.data = data;
        this.prev = null;
        this.next = null;
    }
}
```

Kelas **DoublyLinkedList**
***DoublyLinkedList** class*

```
class DoublyLinkedList {
    Node head;
    Node tail;

    DoublyLinkedList() {
        this.head = null;
        this.tail = null;
    }

    void append(int data) {
         // append method adds a new node with the given data to
the end of the doubly linked list.
        // Your implementation here
    }

    void removeThreeElements(Node startNode) {
        /**
         * Removes three consecutive elements starting from the
given node.
         * If fewer than three elements remain, removes all
remaining elements.
         *
         * @param startNode The starting node from which to
remove elements.
         */
        // Your implementation here
    }

    void display() {
        // Your implementation here
    }
}
```

Dalam kelas DoublyLinkedList, lengkapkan kesemua pelaksanaan untuk TIGA (3) kaedah append, removeThreeElements, dan display. Kaedah removeThreeElements harus mengalih keluar tiga elemen berturut-turut daripada senarai berganda, bermula dari nod startNode. Jika terdapat kurang daripada tiga elemen yang tinggal dalam senarai bermula dari startNode, semua elemen yang tinggal harus dialih keluar.

In the DoublyLinkedList class, complete the implementations for THREE (3) methods append, removeThreeElements, and display. The method removeThreeElements should remove three consecutive elements from the doubly linked list, starting from the node startNode. If there are fewer than three elements remaining in the list starting from startNode, all remaining elements should be removed.

Contoh program ujian dan *output*:
Example test program and output:

```
DoublyLinkedList dll = new DoublyLinkedList();

dll.append(1);
dll.append(2);
dll.append(3);
dll.append(4);
dll.append(5);
dll.append(6);
dll.append(7);
dll.append(8);
dll.append(9);

dll.display(); // Output: 1 2 3 4 5 6 7 8 9
dll.removeThreeElements(dll.head.next); // Remove elements
starting from the second node
dll.display(); // Output: 1 5 6 7 8 9
```

(3 markah/*marks*)

b) Anda ditugaskan untuk melaksanakan program untuk mensimulasikan barisan pelanggan yang menunggu di barisan di bank. Walau bagaimanapun, disebabkan perubahan peraturan baru-baru ini, bank telah memutuskan untuk melaksanakan sistem keutamaan untuk memberi perkhidmatan kepada pelanggan. Keutamaan ditentukan berdasarkan jumlah wang yang ingin disimpan oleh setiap pelanggan.

You are tasked with implementing a program to simulate a queue of customers waiting in line at a bank. However, due to recent changes in regulations, the bank has decided to implement a priority system for serving customers. The priority is determined based on the amount of money each customer intends to deposit.

i) **Pelaksanaan Gilir Keutamaan:** Laksanakan struktur data baris gilir keutamaan dalam Java menggunakan sebarang kaedah yang anda inginkan, dengan memastikan operasi pemasukan dan pengalihan keluar yang cekap. Pastikan pelaksanaan anda menyokong operasi berikut:

Priority Queue Implementation: Implement a priority queue data structure in Java using any method you prefer, ensuring efficient insertion and removal operations. Ensure that your implementation supports the following operations:

- **enqueue(int customerID, double depositAmount):** Tambahkan pelanggan baharu pada baris gilir dengan ID pelanggan dan jumlah deposit yang diberikan. Pelanggan yang mempunyai jumlah deposit yang lebih tinggi harus mempunyai keutamaan yang lebih tinggi.

  enqueue(int customerID, double depositAmount): *Add a new customer to the queue with the given customer ID and deposit amount. Customers with higher deposit amounts should have higher priority.*

- **dequeue(): customer:** Alih keluar dan pulangkan pelanggan dengan keutamaan tertinggi (iaitu, yang mempunyai jumlah deposit tertinggi). Jika bilangan pelanggan mempunyai keutamaan tertinggi yang sama, tolak giliran pelanggan yang tiba dahulu.

  dequeue(): customer: *Remove and return the customer with the highest priority (i.e., the one with the highest deposit amount). If multiple customers have the same highest priority, dequeue the one who arrived first.*

ii) **Pelaksanaan Tindanan:** Laksanakan struktur data tindanan dalam Java. Pelaksanaan anda harus menyokong operasi berikut:

Stack Implementation: Implement a stack data structure in Java. Your implementation should support the following operations:

- **push(Customer element):** Tambahkan pelanggan pada bahagian atas tindanan.

  push(Customer element): *Add a customer to the top of the stack.*

- **pop(): Customer:** Alih keluar dan kembalikan pelanggan dari bahagian atas tindanan. Jika timbunan kosong, kembalikan null.

  pop(): Customer: *Remove and return the customer from the top of the stack. If the stack is empty, return null.*

- **peek(): Customer:** Kembalikan pelanggan di bahagian atas tindanan tanpa mengalihkannya. Jika timbunan kosong, kembalikan null.

  peek(): Customer: *Return the customer at the top of the stack without removing it. If the stack is empty, return null.*

iii) **Pelaksanaan Kelas Pelanggan**: Laksanakan kelas **Pelanggan** dengan atribut dan tingkah laku berikut:

Customer Class Implementation*: Implement a* ***Customer*** *class with the following attributes and behaviors:*

- Ciri-ciri:
  Attributes:

  ✓ **int customerID:** Mewakili ID unik pelanggan.

  int customerID: *Represents the unique ID of the customer.*

  ✓ **double depositAmount:** Mewakili jumlah wang yang pelanggan ingin deposit.

  double depositAmount: *Represents the amount of money the customer intends to deposit.*

- Kelakuan
  Behaviors:

  ✓ Laksanakan pembina untuk memulakan objek Pelanggan dengan ID pelanggan dan jumlah deposit yang diberikan.

  Implement a constructor to initialize a Customer object with the given customer ID and deposit amount.

  ✓ Laksanakan kaedah yang diperlukan untuk membolehkan perbandingan objek Pelanggan berdasarkan jumlah deposit

mereka, membolehkan mereka diisih dalam baris gilir keutamaan.

Implement the necessary methods to enable comparison of Customer objects based on their deposit amounts, allowing them to be sorted in a priority queue.

iv) Reka bentuk dan laksanakan **program Java** untuk mensimulasikan senario berikut:

*Design and implement a **Java program** to simulate the following scenario:*

- Pada mulanya, tiada pelanggan dalam baris gilir bank atau timbunan.

  Initially, there are no customers in the bank queue or stack.

- Menjana 10 pelanggan secara rawak dengan ID pelanggan unik dan jumlah deposit antara RM500 hingga RM10,000.

  Randomly generate 10 customers with unique customer IDs and deposit amounts ranging from RM500 to RM10,000.

- Beratur setiap pelanggan ke dalam baris gilir bank.

  Enqueue each customer into the bank queue.

- Layan setiap pelanggan dengan cara berikut:

  Serve each customer in the following manner:

  ✓ Jika jumlah deposit kurang daripada RM4000, tolak ID pelanggan ke dalam tindanan.

  If the deposit amount is less than RM4000, push the customer onto the stack.

  ✓ Jika jumlah deposit lebih besar daripada atau sama dengan RM4000, tolak baris gilir pelanggan dan cetak ID pelanggan dan jumlah deposit mereka.

  If the deposit amount is greater than or equal to RM4000, dequeue the customer from the queue and print their customer ID and deposit amount.

  ✓ Selepas melayani semua pelanggan, cetak kandungan tindanan, yang mewakili pelanggan yang mempunyai deposit kurang daripada RM4000 tetapi belum dilayan.

  After serving all customers, print the contents of the stack, which represents the customers who had deposits less than RM4000 but were not yet served.

Pastikan pelaksanaan anda ... nya.
Anda mungkin menganggap bahawa ID pelanggan adalah integer unik. Anda dibenarkan untuk menggunakan ekstensi Collections dan Comparator dalaman daripada Java.

Ensure your implementations are efficient and handle edge cases appropriately. You may assume that the customer IDs are unique integers. You are allowed to use the Java internal Collections and Comparator library.

Contoh *output*:
Example output:

```
Customers served:
Customer ID: 7    Deposit: RM9607.46
Customer ID: 8    Deposit: RM9037.85
Customer ID: 1    Deposit: RM8770.78
Customer ID: 5    Deposit: RM8250.10
Customer ID: 2    Deposit: RM7643.66
Customer ID: 4    Deposit: RM5481.05
Customer ID: 3    Deposit: RM5344.36
Customer ID: 9    Deposit: RM4851.38

Customers in stack (deposits < RM4000):
Customer ID: 10    Deposit: RM838.54
Customer ID: 6    Deposit: RM3884.63
```

(7 markah/*marks*)

42.a) Untuk mencetak semua elemen dalam senarai berkait, kaedah lintasan diperlukan untuk mengakses semua nod dalam senarai tersebut. Tuliskan **DUA (2)** kaedah lintasan untuk **senarai berkait tunggal** dan **senarai berkait tunggal berpusar** menggunakan konsep rekursif.

*In order to print all the elements within a linked list, a traversal method is needed to access all nodes in the list. Write **TWO (2)** traversal methods for **singly linked list** and **circular singly linked list** using recursive concept.*

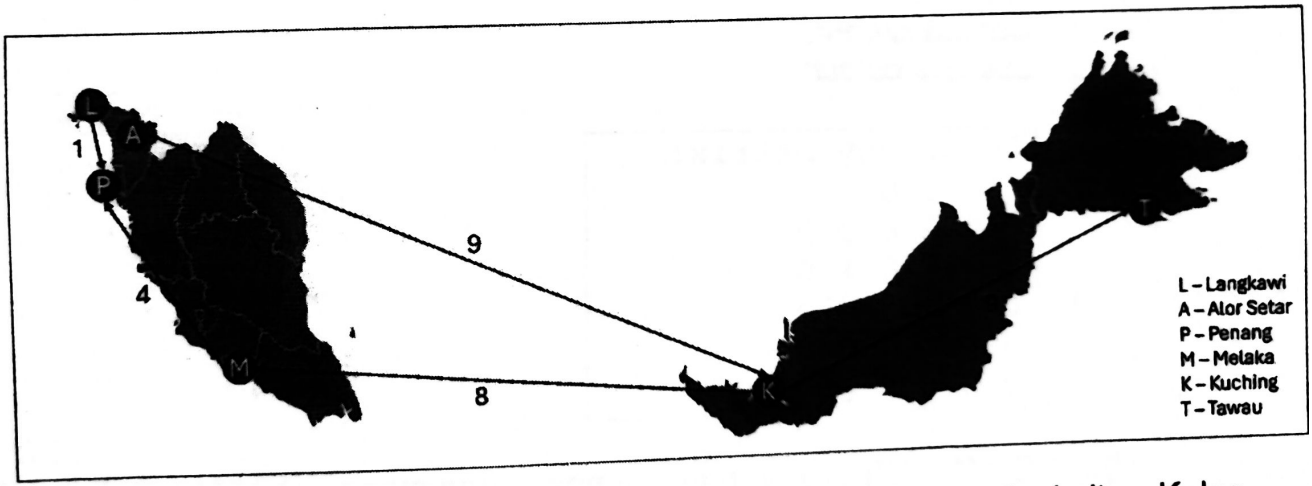
(4 markah/*marks*)

b) Anda diberikan empat fail .java seperti di bawah.
You are given four .java files as below.

- **Vertex.java**
- **Edge.java**
- **WeightedGraph.java**
- **AppGraph.java**

Sebuah **graf1** telah direka berdasarkan gambar rajah berikut, yang menggambarkan penerbangan antara bandar. Bandar-bandar yang terlibat adalah {"Alor Setar", "Kuching", "Langkawi", "Melaka", "Penang", "Tawau"}.

*A **graph1** is created based on the following figure, which illustrates flight between cities. The cities involved are {"Alor Setar", "Kuching", "Langkawi", "Melaka", "Penang", "Tawau"}.*



Vertex untuk **graf1** telah diwujudkan menggunakan Senarai Berkaitan Kelas Vertex dan Edges. Tugasan utama anda di sini adalah untuk menggunakan **graf1** untuk mengekstrak matriks berat kejiranan. Fail **WeightedGraph.java** dilengkapi dengan banyak kaedah yang telah dicipta. Sila gunakan mana-mana kaedah yang disediakan untuk mencapai objektif anda.

*The vertices for **graph1** has been instantiated using Linked List of Vertex and Edges classes. Your main task here is to reverse engineer **graph1** to extract the adjacency weight matrix. The **WeightedGraph.java** file comes with many methods already created. Feel free to use any of the methods provided to achieve your objective.*

Arahan khusus adalah seperti di bawah.
Specific instructions are as below.

i) Tambahkan kaedah ***adjMatrix*** yang mengekstrak dan menyimpan berat tepi dalam Matriks Kejiranan dari graf1 yang dibuat. Anda harus menggunakan array 2 dimensi untuk menyimpan berat. Tetapan baris / lajur matriks adalah seperti berikut:

*Add a method **adjMatrix** which extracts and stores the weights of the edges in an adjacency Matrix from the created graph1. You must use a 2 dimensional array to store the weights. The row/column settings of the matrix are as follow:*

row /column 1: "Alor Setar"
row /column 2: "Kuching"
row /column 3: "Langkawi"
row /column 4: "Melaka"
row /column 5: "Penang"
row /column 6: "Tawau"

ii) Tambahkan kaedah void statik dipanggil **paparAdjMatrix** yang menampilkan matriks kejiranan berat dan papar Matriks Kejiranan sebagai output anda di terminal.

*Add a static void method called **displayAdjMatrix** that displays the weighted adjacency matrix and display the adjacency Matrix as your output in the terminal.*

Contoh *output*:
Example output:

```
Adjacency Matrix:
0 9 0 0 0 0
0 0 0 8 0 9
0 0 0 0 1 0
0 0 0 0 4 0
0 0 0 0 0 0
0 0 0 0 0 0
```

iii) Graf1 telah diubah menjadi pokok dan mengambil "Alor Setar" sebagai akar. Tambahkan kaedah yang dipanggil **traverseTree** dalam **WeightedGraph.java**. Anda harus melintasi dari simpul akar Alor Setar, dalam cara Pencarian Kedalaman Pertama sehingga simpul hujung (bandar). Petua: Anda boleh melintasi graf dengan melaksanakan rekursi pada simpul yang disimpan pada setiap aras pokok.

*Graph1 is modified as tree and is taking "Alor Setar" as root node. Add a method called **traverseTree** in **WeightedGraph.java**. You should traverse from the root node Alor Setar, in the manner of Depth First Search until the leaf node (city). Hint: You can traverse the graph by implementing recursion on the nodes stored at each level of the tree.*

Semasa melintasi, paparkan simpul/bandar pada setiap aras. Anda boleh menambah kaedah **paparPohon** untuk tujuan ini.

When traversing, display the node/cities at each level. You can add a ***displayTree*** *method for this purpose.*

Contoh *output*:
Example output:

```
Tree Traversal:
Traversal: Kuching >
Traversal: Melaka > Tawau >
Traversal: Penang >
Traversal:
End Traversing..
```

(6 markah/*marks*)

43.a) Anda diberikan algoritma carian seperti berikut.

You are given the following search algorithm.

```
public static void main(String[] args) {
        String[] arr = {"Alexa", "Alice", "Alan", "Alvin",
"Alicia", "Aidan", "Ali", "Alicia"};
        String target = "Alicia";
        System.out.println("Target element is: " + target);
        _____ // Question 43.a)(i)
        System.out.println("Sorted array: " +
Arrays.toString(arr));
        List<Integer> result = binarySearch(arr, target);
        if (!result.isEmpty()) {
            System.out.println("Element found at index: " +
result);
        } else
        {    System.out.println("Element not found."); }
    }
```

(Sambungan di muka surat seterusnya)
(Continue next page)

```
public static List<Integer> binarySearch(String[] arr, String
target) {

        List<Integer> indices = new ArrayList<>();

        int left = 0;

        int right = arr.length - 1;

        while (left <= right) {

            int mid = left + (right - left) / 2;

            _______ // Question 43.a)(ii)

            if (comparisonResult == 0) {

                int firstOccurrence = mid;

                while (firstOccurrence > left &&
arr[firstOccurrence - 1].equals(target)) {

                    firstOccurrence--;

                }

                int current = firstOccurrence;

                while (current < arr.length &&
arr[current].equals(target)) {

                    ________ // Question 43.a)(iii)

                    current++;

                }

                return indices;

            } else if (comparisonResult < 0)

            {    left = mid + 1;    }

             else

            {    right = mid - 1;   }

        }

        return indices;  }

}
```

Isi tempat kosong di (i), (ii) dan (iii) untuk mendapatkan output seperti berikut:
Fill in the blanks at (i), (ii) and (iii) to get the following output:

```
Target element is: Alicia
Sorted array: [Aidan, Alan, Alexa, Ali, Alice, Alicia,
Alicia, Alvin]
Element found at index: [5, 6]
```

(3 markah/*marks*)

b) Anda diberi senarai objek **Car**, setiap satu mengandungi tiga atribut: **colour**, **model**, dan **plateNumber**. Laksanakan satu algoritma pokok carian binari (BST) dalam Java yang membina BST daripada senarai objek Car yang diberikan berdasarkan **plateNumbernya**. Selain itu, sediakan satu fungsi carian yang mengambil nombor plat sebagai input dan mengembalikan rentetan yang mengandungi warna dan model kereta yang sepadan dengan nombor plat tersebut. Pastikan operasi pembinaan dan pencarian BST adalah cekap.

*You are given a list of **Car** objects, each containing three fields: **colour**, **model**, and **plateNumber**. Implement a binary search tree (BST) algorithm in Java that constructs a BST from the given list of **Car** objects based on their **plateNumber**. Additionally, provide a **search** function that takes a plate number as input and returns a string containing the colour and model of the car corresponding to that plate number. Ensure that the BST construction and search operations are efficient.*

Pelaksanaan anda hendaklah termasuk yang berikut:
Your implementation should include the following:

i) Satu kelas ***Car*** dengan atribut ***colour***, ***model***, dan ***platNumber*** yang melaksanakan antara muka ***Comparable***.

*A **Car** class with **colour**, **model**, and **plateNumber** fields that implements **Comparable** interface.*

ii) Satu klas ***TreeNode*** yang mewakili nod dalam pepohon carian binari.

*A **TreeNode** class representing a node in the binary search tree.*

iii) Satu metod ***constructBST*** untuk membina pepohon carian binari daripada tatasusunan objek ***Car*** yang diberikan berdasarkan ***plateNumber*** mereka.

*A **constructBST** method to construct a binary search tree from the given array of **Car** objects based on their **plateNumber**.*

iv) Satu metod ***printInAlphabeticalOrder*** untuk mencetak senarai kereta dalam susunan abjad.

*A **printInAlphabeticalOrder** method to print the list of cars in alphabetical order.*

v) Satu metod ***search*** untuk mencari kereta berdasarkan ***plateNumber*** dan mengembalikan rentetan yang mengandungi *colour* dan *model*nya, atau " *Car not found* " jika nombor plat tidak sepadan dengan mana-mana kereta.

*A **search** method to search for a car based on plate number and return a string containing its colour and model, or "Car not found" if the plate number does not match any car.*

Metod main adalah diberikan seperti di bawah (anda tidak boleh mengubahnya):
The main function is given as below (you must not change it):

```
public static void main(String[] args) {
    Car[] cars = {
        new Car("Gotham Gray", "Toyota", "JWY854"),
        new Car("Goodwood Green", "Honda", "KFT9062"),
        new Car("Daybright Blue,", "Proton", "DEX6980"),
        new Car("Polymimetic Gray", "Perodua", "VMG3054"),
        new Car("Gold Mercury", "BMW", "MDU6304"),
        new Car("Sepia Metallic", "Tesla", "CEW5309"),
        new Car("Cyber Yellow", "Audi", "PRE4685"),
        new Car("Panther Black", "Mercedes", "RAU2461")
    };

    BST registry = new BST();

    // Construct the BST
    TreeNode<Car> root = registry.constructBST(cars);

    // Print the list of cars in alphabetical order
    System.out.println("List of the cars in alphabetical
order:");
    registry.printInAlphabeticalOrder(root);

    // Search for a car by plate number and print its colour
and model
    String plateNumberToSearch = "DEX6980";
    String carInfo = registry.search(root,
plateNumberToSearch);
    System.out.println("\nCar information for plate number "
+
        plateNumberToSearch + ": " + carInfo);
    }
}
```

Contoh output:
Example output:

```
List of the cars in alphabetical order:
Car{colour='Sepia Metallic', manufacturer='Tesla',
plateNumber='CEW5309'}
Car{colour='Daybright Blue,', manufacturer='Proton',
plateNumber='DEX6980'}
Car{colour='Gotham Gray', manufacturer='Toyota',
plateNumber='JWY854'}
Car{colour='Goodwood Green', manufacturer='Honda',
plateNumber='KFT9062'}
Car{colour='Gold Mercury', manufacturer='BMW',
plateNumber='MDU6304'}
Car{colour='Cyber Yellow', manufacturer='Audi',
plateNumber='PRE4685'}
Car{colour='Panther Black', manufacturer='Mercedes',
plateNumber='RAU2461'}
Car{colour='Polymimetic Gray', manufacturer='Perodua',
plateNumber='VMG3054'}

Car information for plate number DEX6980: Colour: Daybright
Blue,, Manufacturer: Proton
```

(7 markah/*marks*)

TAMAT
END