# DSA-PROJECT-2024

****

**DATA STUCTURE AND ALGORITHMS ASSIGNMENT: SEMESTER 2 2024**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **SN** | **Name** | **Student Number** | **Role played in the project** | **Mark over 100% (for lecturers use only)** |
| 1 | **Operi R. Kangoortui(TL)** | 221109358 | Pseudocode and Documenting |  |
| 2 | Kavai Chido Angelique | 224009133 | Flowchart and Graphical User Interface |  |
| 3 | Mike Mukungu Katupisa | 223059005 | Pseudocode and Graphical User Interface |  |
| 4 | Undjizuvira U Mengo | 222061634 | Module |  |
| 5 | Shikongo Sabina N | 224080237 | Flowchart |  |
| 6 | Venasiu Nikilas O - | 223090263 | Module |  |

**Explanation of the Module**

1. Contact Management Module

Purpose: Handles operations related to adding, updating, and deleting contacts.

Functions:

Insert Contact (): Add new contact details.

Delete Contact (): Remove a contact from the list.

Update Contact (): Modify existing contact details.

2. Search Module

Purpose: Manages searching through the contact list.

Functions:

Search Contact (): Search for a contact by name or number.

Advanced Search(): If implemented, search using multiple criteria (optional).

3. Display Module

Purpose: Responsible for displaying contact information.

Functions:

Display All Contacts(): Show all contacts in the phonebook.

Sort Contacts(): Sort the contacts alphabetically to make searching faster (optional).

4. Analysis Module (Optional)

Purpose: Analyze the efficiency of the search algorithm.

Functions:

Analyze Search Efficiency(): Analyze the complexity or runtime of the search algorithm used

**FUNCTIONS**

**1. Insert Contact**

FUNCTION InsertContact(phonebook, name, number)

CREATE new\_contact with name and number

ADD new\_contact to phonebook list

END FUNCTION

**2. Search Contact**

FUNCTION SearchContact(phonebook, name)

FOR EACH contact IN phonebook

IF contact.name == name

RETURN contact

END IF

END FOR

RETURN "Contact not found"

END FUNCTION

**3. Display All Contacts**

FUNCTION DisplayAllContacts(phonebook)

IF phonebook is empty

PRINT "No contacts available"

ELSE

FOR EACH contact IN phonebook

PRINT contact.name, contact.number

END FOR

END IF

END FUNCTION

**4. Delete Contact**

FUNCTION DeleteContact(phonebook, name)

FOR EACH contact IN phonebook

IF contact.name == name

REMOVE contact from phonebook

RETURN "Contact deleted"

END IF

END FOR

RETURN "Contact not found"

END FUNCTION

**5. Update Contact**

FUNCTION UpdateContact(phonebook, name, new\_number)

FOR EACH contact IN phonebook

IF contact.name == name

contact.number = new\_number

RETURN "Contact updated"

END IF

END FOR

RETURN "Contact not found"

END FUNCTION

**6. Sort Contacts (optional)**

FUNCTION SortContacts(phonebook)

SORT phonebook by contact.name

END FUNCTION

**Pseudocode:**

" // Binary search is for searching for contact through your contact list

binarySearch (sortedContacts, name)

// Initialize left and right pointers

Initialize left as 0

Initialize right as size of sortedContacts - 1

// Perform binary search

While left <= right

Set mid as left + (right - left) / 2

If sortedContacts[mid].name == name

Return mid // Contact found

Else if sortedContacts[mid].name < name

Set left as mid + 1

Else

Set right as mid - 1

Return -1 // Contact not found

// Is for sorting contacts in a certain format, e.g. alphabetical order.

mergeSort(list)

// If list size is 1 or less, it is already sorted

If list size <= 1

Return list

// Split list into left and right halves

Split list into left and right halves

// Recursively sort each half and merge them

Return merge(mergeSort(left), mergeSort(right))

- merge(left, right)

// Initialize merged as an empty list

Initialize merged as an empty list

// Merge left and right lists

While left and right are not empty

If left[0].name <= right[0].name

Add left[0] to merged

Remove left[0] from left

Else

Add right[0] to merged

Remove right[0] from right

// Add remaining elements of left to merged

Add remaining elements of left to merged

// Add remaining elements of right to merged

Add remaining elements of right to merged

Return merged

// Adding contact

Attributes:

- int size = 100 // Maximum size of the queue

- Contact queue[] = new Contact[size] // Declaring array called queue

- int front = -1 // Setting empty queue

- int rear = -1 // Setting empty queue

Methods:

- enqueue(contact):

IF (rear == size - 1) THEN // Check if the queue is full

DISPLAY "Queue is full"

ELSE IF (front == -1 AND rear == -1) THEN // Check if the queue is empty

front = 0 // Move front to index 0

rear = 0 // Move rear to index 0

queue[rear] = contact // Insert element in queue

ELSE

rear = rear + 1 // Move rear to next index position

queue[rear] = contact // Insert element in queue

ENDIF

- dequeue():

IF (front == -1) THEN // Check if the queue is empty

DISPLAY "Queue is empty"

RETURN NULL

ELSE

contact = queue[front] // Get the front element

IF (front == rear) THEN // Check if the queue has only one element

front = -1 // Reset front

rear = -1 // Reset rear

ELSE

front = front + 1 // Move front to next index position

ENDIF

RETURN contact

ENDIF

- displayContacts():

IF (front == -1) THEN // Check if the queue is empty

DISPLAY "Queue is empty"

ELSE

FOR i = front TO rear DO

DISPLAY queue[i]

ENDFOR

ENDIF

//Deleting contact

- dequeue():

IF (front == -1 AND rear == -1) THEN // Check if the queue is empty

DISPLAY "Queue is empty

ELSE IF (front == rear) THEN

DISPLAY "Deleted element is: " + queue[front]

front = -1 // Reset front

rear = -1 // Reset rear

ELSE

DISPLAY "Deleted element is: " + queue[front]

front = front + 1 // Move front to next index position

ENDIF

// Displaying contact

- displayContacts():

IF (front == -1 AND rear == -1) THEN // Check if the queue is empty

DISPLAY "Queue is empty"

ELSE

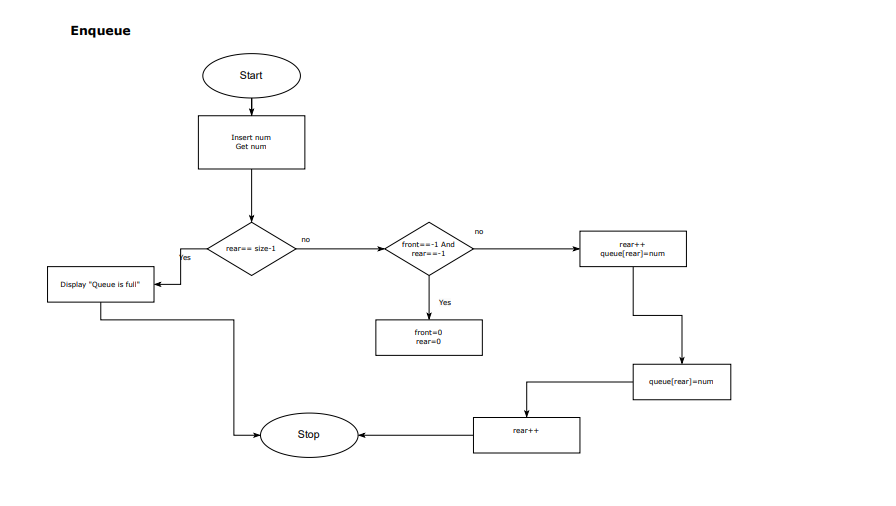
FOR (i = front; i <= rear; i = i + 1) DO

DISPLAY queue[i] // Display elements of queue

ENDFOR

ENDIF

**Flowchart**

****

