Semester Project Spring 2023

Course code: CIS131

Course Title: Data Structure

Submitted to:

Md. Mehedi Hassan

Lecturer, Department of CIS

Daffodil International University

Submitted by:

Md Tamim Hossain

ID: 0242310012091040

Section: B

Department of Computing Information System

Daffodil International University

Table of Contents

Acknowledgment

I would like to express my gratitude to my instructor, mentors, classmates, and family for their unwavering support and guidance during the completion of this assignment on Data structure. Their contributions have been invaluable in enhancing my understanding of the subject matter and enriching my learning experience. Thank you for your assistance and encouragement

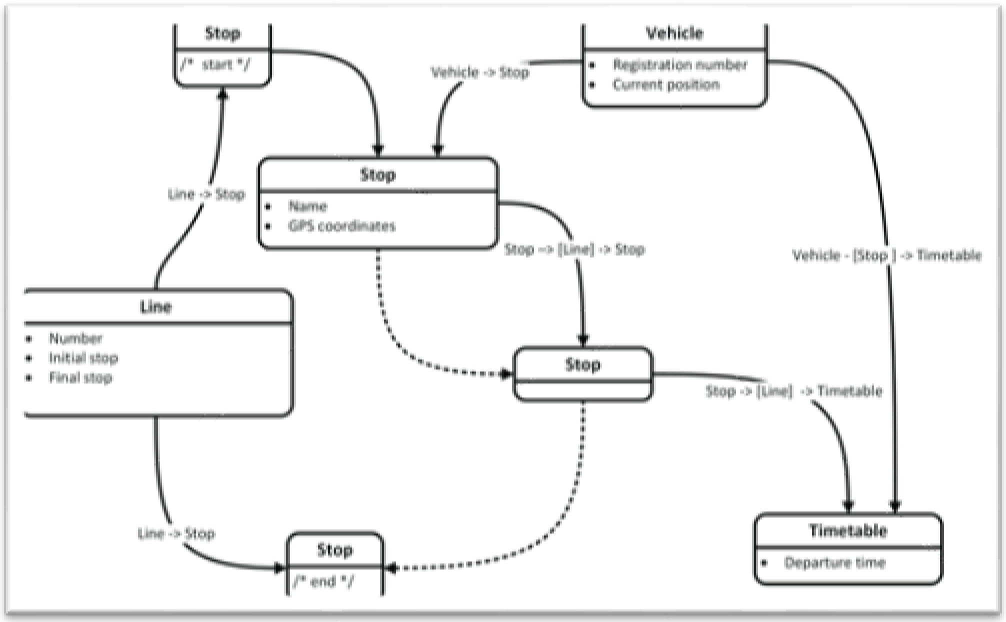
Introduction

Data structures are an essential component of computer science education, as they provide efficient ways to store and organize data. In this assignment, we will explore two scenarios that illustrate the practical applications of data structures. In the first scenario, we will consider three projects proposed by members of the CIS department of Daffodil International University in Bangladesh. These projects involve creating online ticketing systems, library management systems, and web page access tools, all of which require the use of data structures to manage and manipulate information. The second scenario involves a student named Mr. Tusher, who has successfully built a binary search tree but is struggling to create a Max Heap tree. We will examine the differences between these two data structures and explore the challenges and benefits of using them. Through these scenarios, we will gain a deeper understanding of the importance of data structures in computer science , as well as their practical applications in real-world projects.

TASK-1 Answer to Question No: Q1

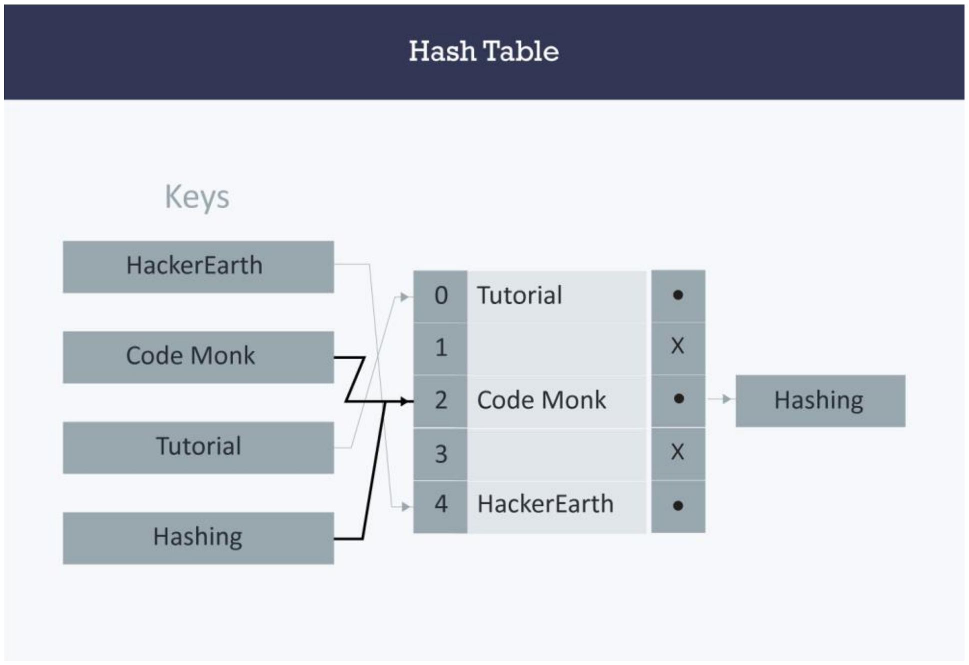
Well, they can use different sorts of data structure. As no specific data structure is mentioned here, I will describe the situation based on the knowledge I have gathered from my course.

Mr. Tamim's project is about buying tickets online, which involves handling data related to bus and train schedules, availability, pricing, and customer information. One possible data structure that could be used in this project is a graph, which can represent the schedules and routes of different buses and trains. Graphs are a common data structure used in transportation and logistics systems.

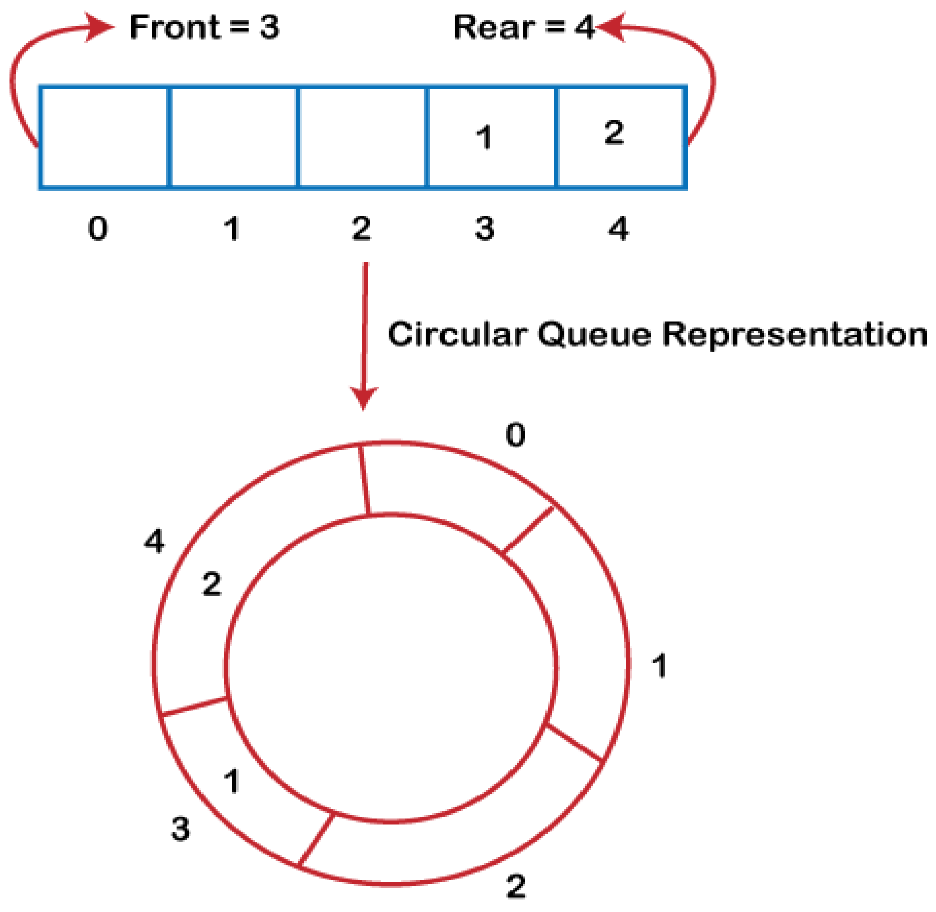


Graph

Mr. Munna's project is about organizing books in a library management system, which involves storing and retrieving information about books such as their titles, authors, publishers, and genres. One possible data structure that could be used in this project is a hash table, which can efficiently store and retrieve key-value pairs. Hash tables are a common data structure used in database systems



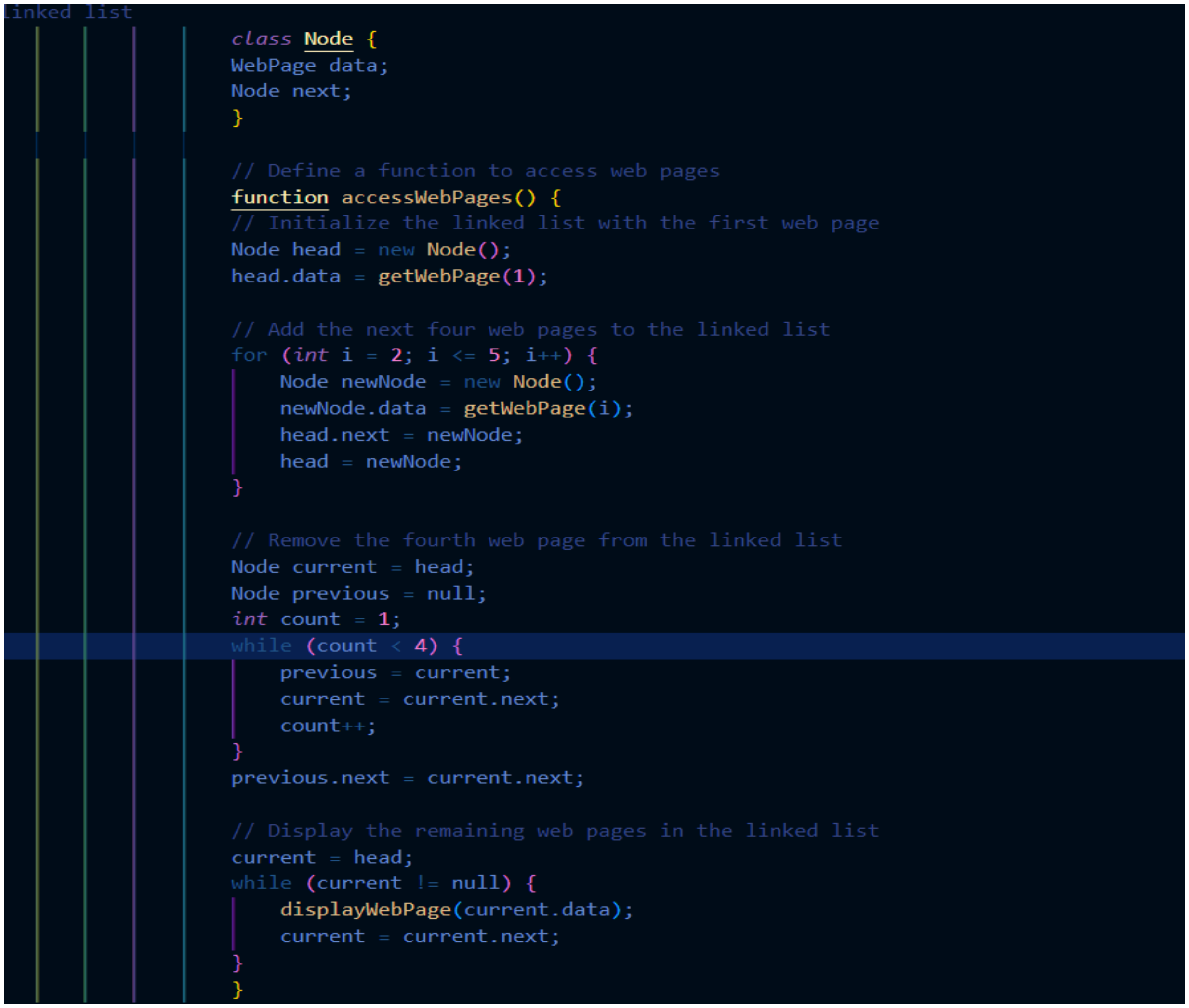
Regarding **Mr. Naim**'s scenario, he executed a queue of characters, where QUEUE is a circular array with a fixed size of six memory cells. A queue is a type of data structure that follows the First-In-First-Out (FIFO) principle, where the first item added to the queue is the first item to be removed. In this case, the queue is implemented using a circular array, which means that the front and rear pointers wrap around to the beginning of the array when they reach the end. This can be an efficient way to implement a queue when the number of elements is fixed and known in advance



In summary, the three students in scenario-1 may have used different types of data structures depending on their project requirements. Possible data structures that could be used in their projects include graphs, hash tables, and trees. Additionally, Mr. Naim executed a queue of characters using a circular array.

TASK-1 Answer to Question No: Q2

To simulate Mr. Rimon's web page access scenario using pseudocode, we can use a linked list data structure to represent the sequence of web pages he visits. Here's an example pseudocode implementation



```
linked list
class Node {
WebPage data;
Node next;
}

// Define a function to access web pages
function accessWebPages() {
// Initialize the linked list with the first web page
Node head = new Node();
head.data = getWebPage(1);

// Add the next four web pages to the linked list
for (int i = 2; i <= 5; i++) {
    Node newNode = new Node();
    newNode.data = getWebPage(i);
    head.next = newNode;
    head = newNode;
}

// Remove the fourth web page from the linked list
Node current = head;
Node previous = null;
int count = 1;
while (count < 4) {
    previous = current;
    current = current.next;
    count++;
}
previous.next = current.next;

// Display the remaining web pages in the linked list
current = head;
while (current != null) {
    displayWebPage(current.data);
    current = current.next;
}
}
```

In this pseudocode, the Node class represents a node in the linked list, which contains a WebPage object and a reference to the next node in the list. The accessWebPages() function initializes the linked list with the first five web pages, then removes the fourth web page from the list using a simple traversal algorithm. Finally, the remaining web pages are displayed in the order they appear in the linked list.

Note that this is just one possible implementation of Mr. Rimon's scenario using pseudocode, and there are many other ways to represent and manipulate linked lists in code.

TASK-1 Answer to Question No: Q3

Here's how the queue would look like after each operation:

a) F is added to the queue.

FRONT = 2, REAR = 5, QUEUE: —--, A, C, D, —---, F

b) Two letters are deleted.

FRONT = 4, REAR = 5, QUEUE: —--, A, —---, —---, —---, F

c) K, L, and M are added to the queue.

FRONT = 4, REAR = 1, QUEUE: K, L, M, —---, —---, F

d) Two letters are deleted.

FRONT = 1, REAR = 1, QUEUE: K, —---, —---, —---, —---, F

e) R is added to the queue.

FRONT = 1, REAR = 2, QUEUE: K, R, —---, —---, —---, F

f) Two letters are deleted.

FRONT = 3, REAR = 2, QUEUE: K, R, —---, —---, —---, —---

g) S is added to the queue.

FRONT = 3, REAR = 3, QUEUE: K, R, S, —---, —---, —---

h) Two letters are deleted

FRONT = 4, REAR = 3, QUEUE: K, R, S, —---, —---, —---

i) One letter is deleted

FRONT = 5, REAR = 3, QUEUE: K, R, S, —---, —---, —---

j) One letter is deleted.

FRONT = 0, REAR = 3, QUEUE: K, R, S, —---, —---, —---

In the above queue, FRONT and REAR represent the indices of the front and rear of the queue, respectively. The letters in the QUEUE array represent the contents of the queue. Empty memory cells are denoted by "-----". As operations are performed on the queue, the FRONT and REAR indices change, and items are added to or removed from the QUEUE array accordingly.

TASK-2 Answer to Question No: Q4

To insert data into an empty hash table of size 15 with hash function H(k) = k%15, we can follow the steps below:

Initialize an empty hash table of size 15.

For each data element k, calculate its hash value using the hash function H(k) = k%15.

Check if the hash table has an empty slot at the hash value index.

If the slot is empty, insert the data element k into that slot.

If the slot is not empty, use linear probing to find the next empty slot and insert the data element k there.

Repeat steps 3-5 for all data elements.

Using this process, let's insert the given data (25, 59, 19, 34, 44, 29, 38, 46, 8, 17, 22) into the hash table:

Hash value of 25: 25%15 = 10

Hash value of 59: 59%15 = 14

Hash value of 19: 19%15 = 4

Hash value of 34: 34%15 = 4 (collision, linear probing used)

Hash value of 44: 44%15 = 14 (collision, linear probing used)

Hash value of 29: 29%15 = 14 (collision, linear probing used)

Hash value of 38: 38%15 = 8

Hash value of 46: 46%15 = 1

Hash value of 8: 8%15 = 8 (collision, linear probing used)

Hash value of 17: 17%15 = 2

Hash value of 22: 22%15 = 7

The final output will be the hash table with the data elements inserted:

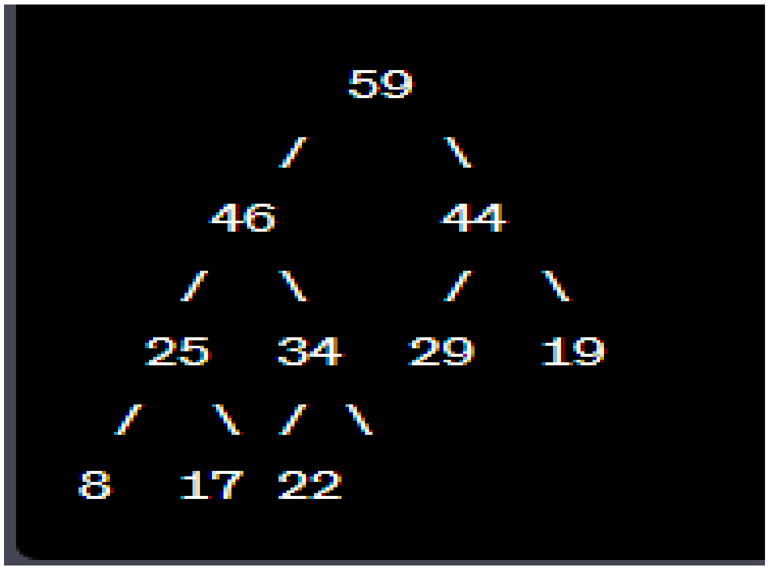
```
Index 0:
Index 1: 46
Index 2: 17
Index 3:
Index 4: 19 34 44
Index 5:
Index 6:
Index 7: 22
Index 8: 8
Index 9:
Index 10: 25
Index 11:
Index 12:
Index 13:
Index 14: 59 29 38
```

Therefore, the final output of the hash table with the given data elements inserted will be as shown above.

TASK-2 Answer to Question No: Q5

To visualize the Max. Heap tree for the given data (25, 59, 19, 34, 44, 29, 38, 46, 8, 17, 22), we need to first build the tree by following the Max Heap property which states that the parent node should always be greater than its child nodes.

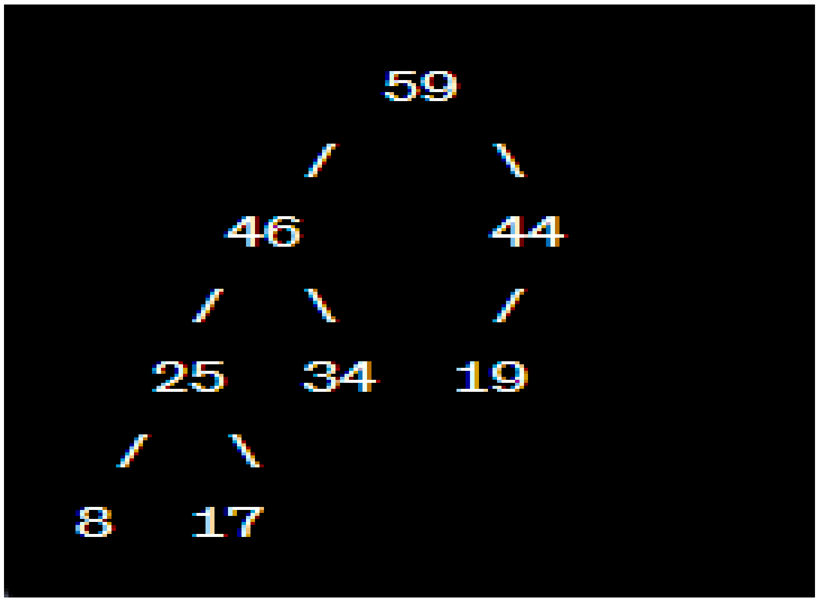
The initial Max Heap tree for the given data would look like this:



```
          59
        /    \
      46     44
     /  \    /  \
    25  34  29  19
   /  \ / \
  8  17 22
```

To delete 29 from the above tree, we replace 29 with the last node of the tree, which is 22 in this case. After that, we compare the node with its children, and swap with the larger child until the Max Heap property is restored.

The reconstructed tree after deleting 29 would look like this:



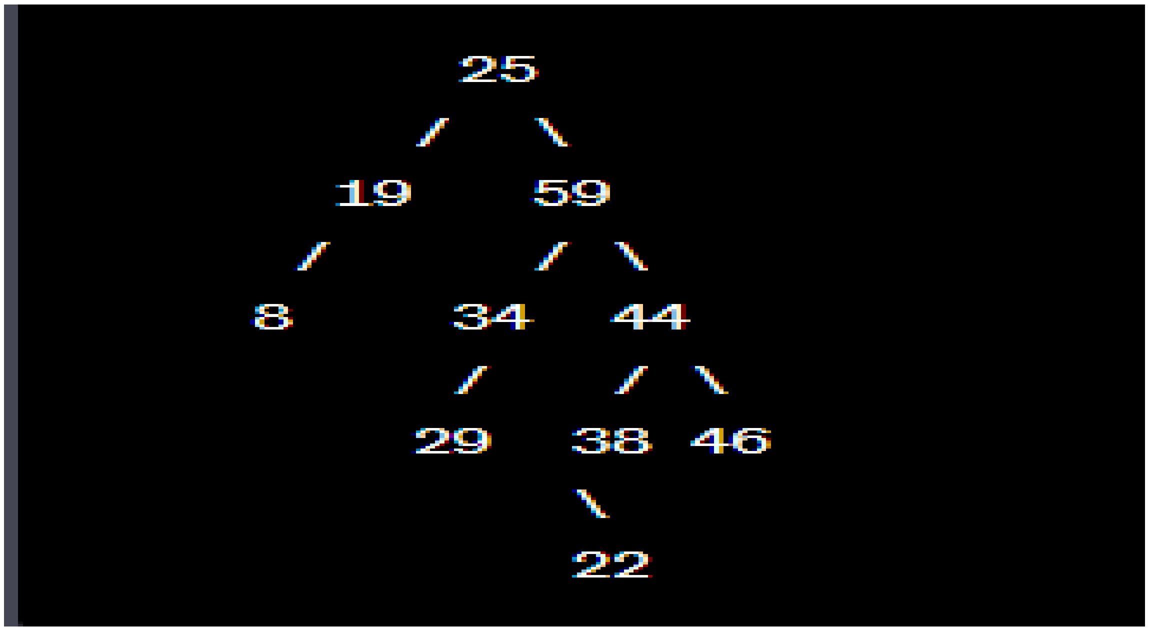
```
        59
      /    \
    46     44
   /  \    /
  25  34  19
 /  \
8   17
```

TASK-2 Answer to Question No: Q6

In a binary search tree, each node has at most two children: a left child and a right child. Moreover, for every node in the tree, all the elements in the left subtree are less than the node, and all the elements in the right subtree are greater than the node. This property is known as the binary search property.

The binary search property makes binary search possible on a BST. Binary search is a search algorithm that works on sorted data structures, and it starts by comparing the search key with the root node of the tree. If the key is less than the root, the algorithm proceeds to the left subtree; otherwise, it proceeds to the right subtree. This process continues until the key is found or there are no more nodes to search. Since the data in a BST is sorted according to the binary search property, binary search can be applied to a BST to efficiently search for a particular element

Based on the data given in Scenario-2, Mr. Tusher built the following binary search tree (BST):



```
     25
    /  \
  19    59
 /      / \
8    34   44
     /    / \
    29   38 46
         \
         22
```

Lab Part

Task – 01

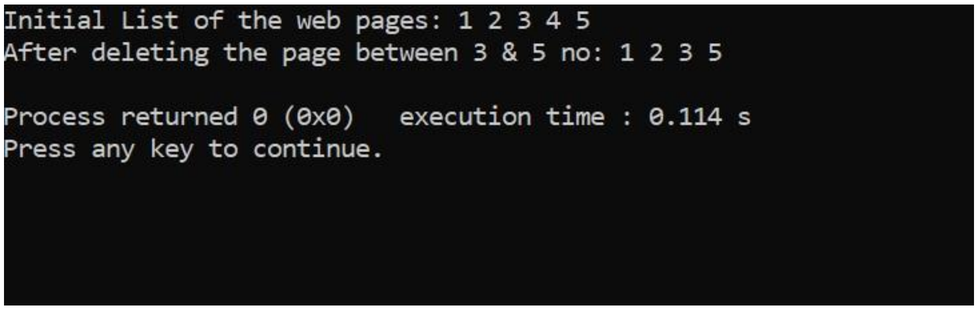
Answer to Q. No – 01

Here is the C++ program to implement the scenario described in Q2:

```cpp
#include <iostream>
using namespace std;

// Define the node structure for our linked list
struct Node
{
    int data;
    Node *next;
};

// Define the linked list class
class LinkedList
{
private:
    Node *head;

public:
    // Constructor to initialize an empty linked list
    LinkedList()
    {
        head = NULL;
    }

    // Method to add a new node to the linked list
    void addNode(int newData)
    {
        Node *newNode = new Node;
        newNode->data = newData;
        newNode->next = NULL;

        if (head == NULL)
        {
            head = newNode;
        }
        else
        {
            Node *currNode = head;
            while (currNode->next != NULL)
            {
                currNode = currNode->next;
            }
            currNode->next = newNode;
        }
    }

    // Method to delete a node from the linked list
    void deleteNode(int targetData)
    {
        if (head == NULL)
        {
            return;
        }

        if (head->data == targetData)
        {
            Node *tempNode = head;
            head = head->next;
            delete tempNode;
            return;
        }

        Node *currNode = head;
        while (currNode->next != NULL)
        {
            if (currNode->next->data == targetData)
            {
                Node *tempNode = currNode->next;
                currNode->next = tempNode->next;
                delete tempNode;
                return;
            }
            currNode = currNode->next;
        }
    }

    // Method to print the contents of the linked list
    void printList()
    {
        Node *currNode = head;
        while (currNode != NULL)
        {
            cout << currNode->data << " ";
            currNode = currNode->next;
        }
        cout << endl;
    }
};
```

```cpp

int main()
{
    LinkedList webPages;

    webPages.addNode(1);
    webPages.addNode(2);
    webPages.addNode(3);
    webPages.addNode(4);
    webPages.addNode(5);

    cout << "Initial List of the web pages: ";
    webPages.printList();

    webPages.deleteNode(4);
    cout << "After deleting the page between 3 & 5 no: ";
    webPages.printList();

    return 0;
}
```

Output:



```
Initial List of the web pages: 1 2 3 4 5
After deleting the page between 3 & 5 no: 1 2 3 5

Process returned 0 (0x0)   execution time : 0.114 s
Press any key to continue.
```

Answer to Q. No – 02

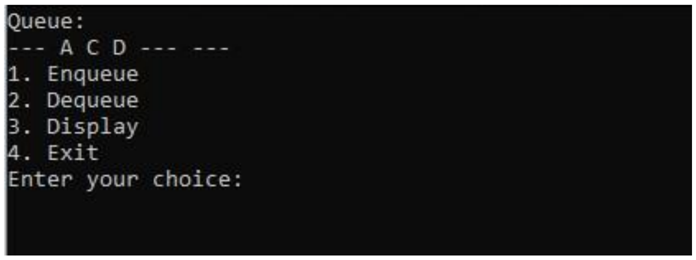
The implementation of the described queue operations in C++:

```cpp
#include <bits/stdc++.h>

using namespace std;

#define CAPACITY 6

char elements[CAPACITY];

int front = 0, rear = -1, total_items = 0;

bool is_full()
{
    if (total_items == CAPACITY)
        return true;
    else
        return false;
}

bool is_null()
{
    if (total_items == 0)
        return true;
    else
        return false;
}

void enqueue()
{
    if (is_full())
    {
        cout << "Queue is full\n";
    }
    else
    {
        char v;
        system("cls");
        cout << "Enter the element: ";
        cin >> v;
        rear = (rear + 1) % CAPACITY;
        elements[rear] = v;
        total_items++;
        cout << "Element added successfully\n";
    }
}

```

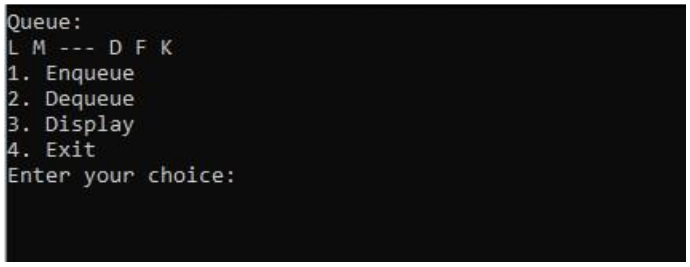
```cpp
void dequeue()
{
    if (is_null())
    {
        cout << "Empty queue\n";
    }
    else
    {
        system("cls");
        elements[front] = NULL;
        front = (front + 1) % CAPACITY;
        total_items--;
        cout << "Dequeued successfully\n";
    }
}

void print_queue()
{
    if (is_null())
    {
        cout << "Queue is empty\n";
        for (int i = 0; i < CAPACITY; i++)
        {
            if (elements[i] == NULL)
                cout << "--- ";
            else
                cout << elements[i] << " ";
        }
        cout << "\n";
    }
    else
    {
        system("cls");
        cout << "Queue:\n";
        for (int i = 0; i < CAPACITY; i++)
        {
            if (elements[i] == NULL)
                cout << "--- ";
            else
                cout << elements[i] << " ";
        }
        cout << "\n";
    }
}

int main()
{
    int input;

    while (1)
    {

```

```
    menu:
        cout << "1. Enqueue\n2. Dequeue\n3. Display\n4. Exit\nEnter your choice: ";
        cin >> input;
        if (input == 1)
            enqueue();
        else if (input == 2)
            dequeue();
        else if (input == 3)
            print_queue();
        else if (input == 4)
            break;
        else
        {
            cout << "Invalid input, please enter again\n";
            goto menu;
        }
    }

    return 0;
}
```

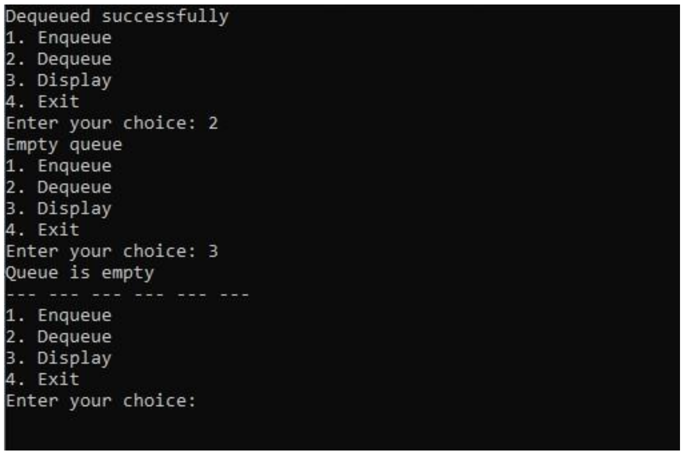
Output:



```
Queue:
--- A C D --- ---
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice:
```



```
Queue:
L M --- D F K
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice:
```



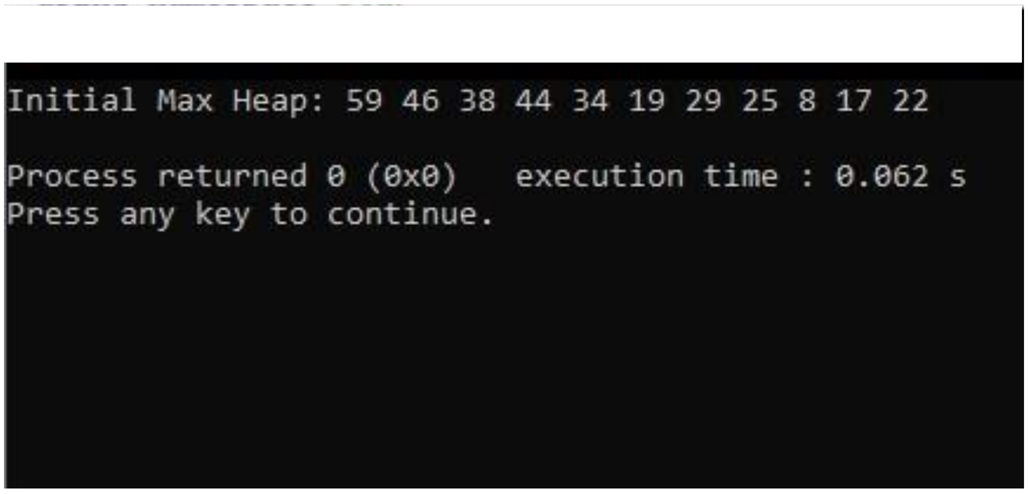
```
Dequeued successfully
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Empty queue
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue is empty
--- --- --- --- --- ---
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice:
```

Answer to Q. No – 03

Implementation of a Max Heap in C++ based on the data provided in the scenario:

```cpp
#include <iostream>
#include <vector>
using namespace std;

class MaxHeap
{
private:
    vector<int> heap;

    // Helper function to swap two elements in the heap
    void swap(int &a, int &b)
    {
        int temp = a;
        a = b;
        b = temp;
    }

public:
    // Constructor to create an empty heap
    MaxHeap() {}

    // Function to insert a new element into the heap
    void insert(int val)
    {
        // Add the new element to the end of the heap
        heap.push_back(val);

        // Move the element up the heap until it's in the correct position
        int index = heap.size() - 1;
        while (index > 0 && heap[(index - 1) / 2] < heap[index])
        {
            swap(heap[index], heap[(index - 1) / 2]);
            index = (index - 1) / 2;
        }
    }

    // Function to print the contents of the heap
    void print()
    {
        for (int i = 0; i < heap.size(); i++)
        {
            cout << heap[i] << " ";
        }
        cout << endl;
    }
};

int main()
{
    // Create a new Max Heap
    MaxHeap heap;

    // Perform the operations mentioned in the scenario
    heap.insert(25);
    heap.insert(59);
    heap.insert(19);
    heap.insert(34);
    heap.insert(44);
    heap.insert(29);
    heap.insert(38);
    heap.insert(46);
    heap.insert(8);
    heap.insert(17);
    heap.insert(22);

    // Print the initial contents of the heap
    cout << "Initial Max Heap: ";
    heap.print();

    return 0;
}
```

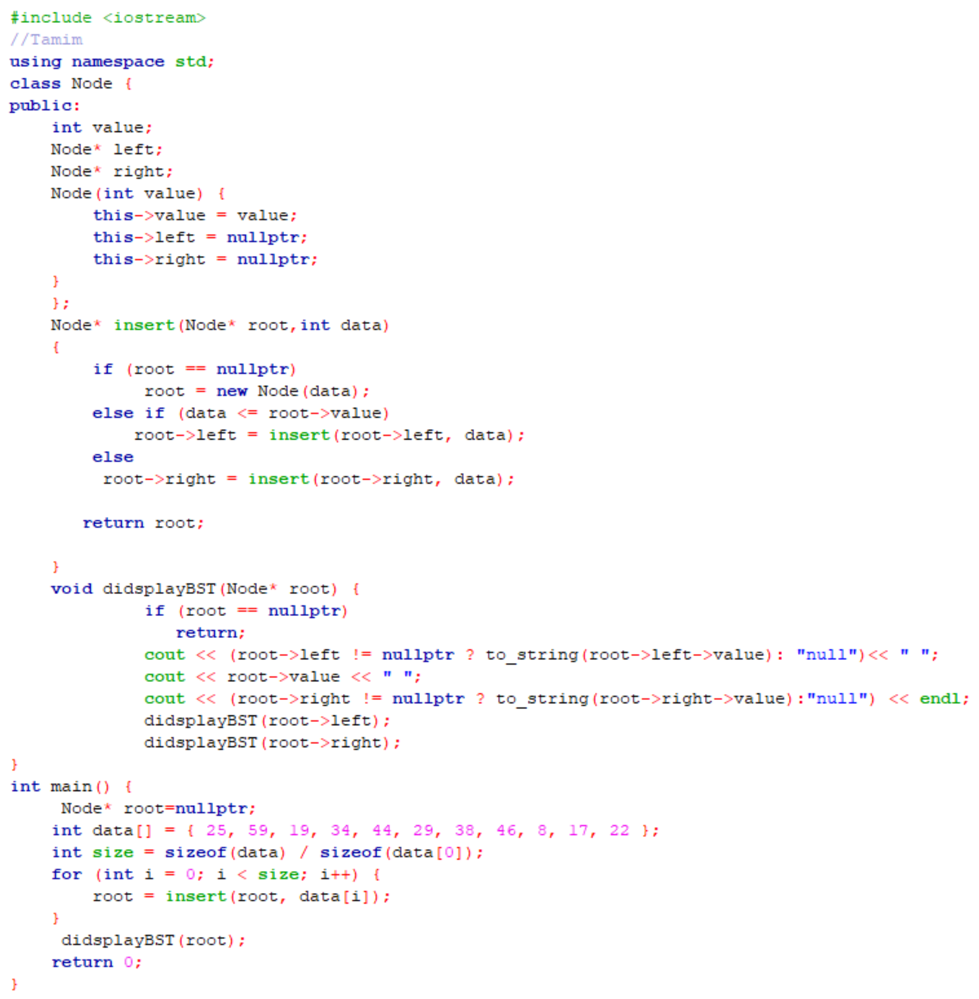
Output



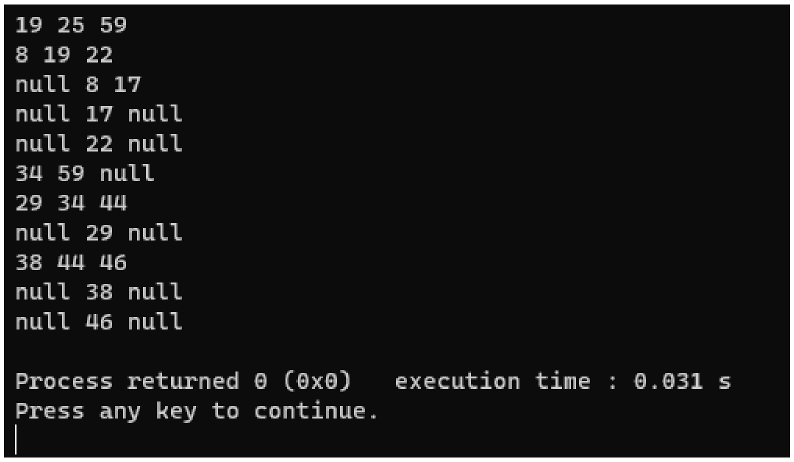
```
Initial Max Heap: 59 46 38 44 34 19 29 25 8 17 22

Process returned 0 (0x0)   execution time : 0.062 s
Press any key to continue.
```

Answer to Q. No – 04

Implementation of the binary search tree (BST) mentioned in Scenario-2 using C++:



```cpp
#include <iostream>
//Tamim
using namespace std;
class Node {
public:
    int value;
    Node* left;
    Node* right;
    Node(int value) {
        this->value = value;
        this->left = nullptr;
        this->right = nullptr;
    }
    };
    Node* insert(Node* root,int data)
    {
        if (root == nullptr)
             root = new Node(data);
        else if (data <= root->value)
            root->left = insert(root->left, data);
        else
         root->right = insert(root->right, data);

       return root;

    }
    void didsplayBST(Node* root) {
             if (root == nullptr)
                return;
             cout << (root->left != nullptr ? to_string(root->left->value): "null")<< " ";
             cout << root->value << " ";
             cout << (root->right != nullptr ? to_string(root->right->value):"null") << endl;
             didsplayBST(root->left);
             didsplayBST(root->right);
}
int main() {
     Node* root=nullptr;
    int data[] = { 25, 59, 19, 34, 44, 29, 38, 46, 8, 17, 22 };
    int size = sizeof(data) / sizeof(data[0]);
    for (int i = 0; i < size; i++) {
        root = insert(root, data[i]);
    }
     didsplayBST(root);
    return 0;
}
```



```
19 25 59
8 19 22
null 8 17
null 17 null
null 22 null
34 59 null
29 34 44
null 29 null
38 44 46
null 38 null
null 46 null

Process returned 0 (0x0)   execution time : 0.031 s
Press any key to continue.
```

Conclusion

In conclusion, data structures play a crucial role in computer science and engineering. Through the two scenarios presented in this assignment, we have seen how data structures can be applied to real-world problems, such as creating online ticketing systems, library management systems, and web page access tools. These projects require the use of data structures to efficiently store and manipulate information, leading to more effective and user-friendly systems. We have also seen the challenges that can arise when working with different types of data structures, such as building a Max Heap tree, which can be difficult but ultimately rewarding. As we continue to develop new technologies and tackle complex problems, data structures will remain a critical component of our toolbox. By mastering these structures and understanding their applications, we can build more robust and efficient systems that meet the needs of today's world