DSA ONLINE 1

Imagine a bank with a lengthy queue of people patiently waiting for their desired services. When a new person arrives, they are added to the line based on their priority. Unfortunately, the bank only has one counter, but the company occasionally opens an emergency counter to minimize customer wait times. During this time, individuals who have requested emergency services are moved from the general queue to the emergency counter, making them special users.

As a skilled engineer hired to tackle this challenge, your task is to develop a C++ program that can perform the following actions. Your program should have the following menu options as numbered below.

1. Add a person to the general user list based on priority.
2. Provide service from the general counter list.
3. Provide service from the emergency counter list.
4. Open the emergency counter.
5. View the current queue, which is in front of the general counter.
6. View the current queue, which is in front of the emergency counter.

You will be provided with three pieces of information for each customer: CustomerID, Priority, and EmergencyService Status. Using a singly linked list, and dynamic array, your program will automate the banking process outlined above. Write separate programs for the solution using Dynamic Array List (DAL) and then Singly Linked Lists. :

For the sample input-output table below, consider the general queue already has the following entries, while the emergency queue has no entries.

90 0 R -> 8 1 E -> 7 4 R -> 6 5 E -> 5 7 E -> 4 8 R -> 3 9 R

Here, in each node, the first field denotes the ID of the customer, followed by the second field denoting the priority number. of the customer, while the third field denotes a status (whether the user applied for

emergency service) where E means emergency and R means regular. New customers are always entered into the general counter before they are moved to the emergency counter (by pressing menu option ‘4’).

Note: The user having the lowest priority value will receive the highest preference.

Sample Input - Output

```
Banking Queue Options:
1. Adding to the general list
2. Provide service from the general
counter list.
3. Provide service from the special
counter list.
4. Move to emergency counter.
5. View general counter
6. View emergency counter
Enter Option:6
General Counter: 90, 8, 7, 6, 5, 4, 3
<<Banking Queue Options: ... >>
Enter Option: 4
Entry entered to emergency counter: 8, 6, 5
<<Banking Queue Options: ... >>
Enter Option: 2
Service provided to ID 90
<<Banking Queue Options: ... >>
Enter Option: 3
Service provided to ID 8
<<Banking Queue Options: ... >>
Enter Option:1
Enter the ID, priority, and emergency status of
the person: 10, 0, E
```

```
ID-10 added to position 1 of the general list

<<Banking Queue Options: ... >>

Enter Option: 5

General Counter: 10, 7, 4, 3

<<Banking Queue Options: ... >>

Enter option: 4

Entry entered to emergency counter: 10

<<Banking Queue Options: ... >>

Enter option: 6

Emergency Counter: 10, 8, 6, 5
```

SOLN:

```
#include <iostream>

class Node {
public:
    int customerID;
    int priority;
    char status; // 'E' for emergency, 'R' for regular
    Node* next;

    Node(int id, int p, char s) : customerID(id), priority(p), status(s), next(nullptr) {}
};

class Queue {
private:
    Node* front;

public:
```

```
Queue() : front(nullptr) {}

void enqueue(int id, int priority, char status) {
    Node* newNode = new Node(id, priority, status);
    if (front == nullptr) {
        front = newNode;
    } else {
        Node* current = front;
        while (current->next != nullptr) {
            current = current->next;
        }
        current->next = newNode;
    }
}

void dequeue() {
    if (front != nullptr) {
        Node* temp = front;
        front = front->next;
        delete temp;
    }
}

void display() {
    Node* current = front;
    while (current != nullptr) {
        std::cout << current->customerID << " ";
        current = current->next;
    }
```

```
        std::cout << std::endl;
    }
};

int main() {
    Queue generalCounterQueue;
    Queue emergencyCounterQueue;

    int option;
    int id, priority;
    char status;

    do {
        std::cout << "Banking Queue Options:" << std::endl;
        std::cout << "1. Adding to the general list" << std::endl;
        std::cout << "2. Provide service from the general counter list" << std::endl;
        std::cout << "3. Provide service from the emergency counter list" << std::endl;
        std::cout << "4. Move to emergency counter" << std::endl;
        std::cout << "5. View general counter" << std::endl;
        std::cout << "6. View emergency counter" << std::endl;
        std::cout << "Enter Option: ";
        std::cin >> option;

        switch (option) {
            case 1:
                std::cout << "Enter the ID, priority, and emergency status of the person: ";
                std::cin >> id >> priority >> status;
                generalCounterQueue.enqueue(id, priority, status);
                std::cout << "ID-" << id << " added to position " << priority << " of the general list" << std::endl;
                break;
            case 2:
                if (generalCounterQueue.front != nullptr) {
                  std::cout << "Service provided to ID " << generalCounterQueue.front->customerID << 
std::endl;
                  generalCounterQueue.dequeue();
                } else {
                  std::cout << "General counter is empty" << std::endl;
                }
                break;
            case 3:
                if (emergencyCounterQueue.front != nullptr) {
                  std::cout << "Service provided to ID " << emergencyCounterQueue.front->customerID << 
std::endl;
                  emergencyCounterQueue.dequeue();
                } else {
                  std::cout << "Emergency counter is empty" << std::endl;
                }
                break;
            case 4:
                Node* current = generalCounterQueue.front;
                while (current != nullptr) {
                  if (current->status == 'E') {
                    int id = current->customerID;
                    char status = current->status;
                    generalCounterQueue.dequeue();
                    emergencyCounterQueue.enqueue(id, priority, status);
                    std::cout << "Entry entered to emergency counter: " << id << std::endl;
                  }
```

```
                current = current->next;
              }
              break;
          case 5:
              std::cout << "General Counter: ";
              generalCounterQueue.display();
              break;
          case 6:
              std::cout << "Emergency Counter: ";
              emergencyCounterQueue.display();
              break;
          default:
              std::cout << "Invalid option" << std::endl;
              break;
        }
    } while (option != 0);

    return 0;
}
```

King Arthur has planned for a round table conference with his many knights. However, the size of the round table is only Five, i.e., only four knights, along with King Arthur, can sit. For the fifth knight to sit at the round table, it was ordered the fifth knight would sit at the first knight position, as the first knight is the first one who came and sat. Similarly, the sixth knight would sit in the second knight's place. So, the fifth and consecutive knights will replace the previous ones sequentially. King Arthur would sit whenever he came in. However, no knight shall replace his position in the round table. Note that 'A' means King Arthur; other knights may have any other letter for their id.

Now, Solve the above Scenario using an appropriate data structure. You should be able to insert and view the current members of the round table. Follow the sample input-output table given below.

Sample Input-Output Table

Input

```
Enter Member Name (first letter): D
```

Output

```
Inserted. Current Round Table: D
```

Input

```
Enter Member Name (first letter): R
```

Output

```
Inserted. Current Round Table: D R
```

Input

```
Enter Member Name (first letter): A
```

Output

```
Inserted. Current Round Table: D R A
```

Input

```
Enter Member Name (first letter): G
```

Output

```
Inserted. Current Round Table: D R A G
```

Input

```
Enter Member Name (first letter): O
```

Output

```
Inserted. Current Round Table: D R A G O
```

Input

```
Enter Member Name (first letter): N
```

Output

```
Table filled, member replaced.
Current Round Table: N R A G O
```

SOLN:

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

const int MAX_SIZE = 5;

class CircularQueue {
private:
    char queue[MAX_SIZE];
    int front, rear;
    int itemCount;

public:
    CircularQueue() : front(-1), rear(-1), itemCount(0) {}

    bool isEmpty() {
        return itemCount == 0;
    }

    bool isFull() {
        return itemCount == MAX_SIZE;
    }

    void enqueue(char member) {
        if (isFull()) {
            cout << "Table filled, member replaced." << endl;
            dequeue();
        }
        if (isEmpty()) {
            front = rear = 0;
        } else {
            rear = (rear + 1) % MAX_SIZE;
        }

        queue[rear] = member;
        itemCount++;
    }

    void dequeue() {
        if (!isEmpty()) {
            front = (front + 1) % MAX_SIZE;
            itemCount--;
        }
    }

    void display() {
        int i = front;
        int count = 0;

        cout << "Current Round Table: ";
        while (count < itemCount) {
            cout << queue[i] << " ";
            i = (i + 1) % MAX_SIZE;
            count++;
        }
        cout << endl;
    }
```

```
};

int main() {
    CircularQueue roundTable;
    char member;

    while (true) {
        cout << "Enter Member Name (first letter): ";
        cin >> member;

        roundTable.enqueue(member);
        roundTable.display();
    }

    return 0;
}
```

Practice Problem Set

Formulate a program to take two linked lists as input and make a new link list with the average value of each index of those two linked lists.

Sample Input

Size of the list 1: 3

Items in List 1: 1 2 3

Size of List 2: 3

Items in List 2: 1 2 3

Sample Output

Output: 1 2 3

Sample Input

Size of the list 1: 5

Items in List 1: 10 20 30 40 50

Size of List 2: 6

Items in List 2: 20 40 60 80 100 120

Sample Output

Output: 15 30 45 60 75 60

SOLN:

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

// Node structure to represent each element of the linked list
struct Node {
    int data;
    Node* next;

    Node(int val) : data(val), next(nullptr) {}
};
```

```
// Function to insert a new node at the end of the linked list
void insert(Node*& head, int val) {
    Node* newNode = new Node(val);
    if (!head) {
        head = newNode;
    } else {
        Node* temp = head;
        while (temp->next) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
}

// Function to calculate the average of two linked lists and create a new linked list with the average
values
Node* calculateAverage(Node* list1, Node* list2) {
    Node* result = nullptr;
    Node* temp1 = list1;
    Node* temp2 = list2;

    while (temp1 && temp2) {
        int avg = (temp1->data + temp2->data) / 2;
        insert(result, avg);
        temp1 = temp1->next;
        temp2 = temp2->next;
    }

    return result;
```

```
}

// Function to display the elements of a linked list
void displayList(Node* head) {
    Node* temp = head;
    while (temp) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}

int main() {
    Node* list1 = nullptr;
    Node* list2 = nullptr;
    int size1, size2;

    // Input for the first linked list
    cout << "Size of the list 1: ";
    cin >> size1;
    cout << "Items in List 1: ";
    for (int i = 0; i < size1; i++) {
        int val;
        cin >> val;
        insert(list1, val);
    }

    // Input for the second linked list
    cout << "Size of the list 2: ";
```

```
cin >> size2;
cout << "Items in List 2: ";
for (int i = 0; i < size2; i++) {
    int val;
    cin >> val;
    insert(list2, val);
}

// Calculate the average of two linked lists and display the result
Node* averageList = calculateAverage(list1, list2);
cout << "Output: ";
displayList(averageList);

// Free the memory occupied by the linked lists
while (list1) {
    Node* temp = list1;
    list1 = list1->next;
    delete temp;
}
while (list2) {
    Node* temp = list2;
    list2 = list2->next;
    delete temp;
}
while (averageList) {
    Node* temp = averageList;
    averageList = averageList->next;
    delete temp;
}
```

```
    return 0;
}
```

Formulate a program to take two linked lists and merge them in another linked list in sorted order.

Sample Input

Size of the list 1: 5

Items in List 1: 1 9 2 4 10

Size of List 2: 3

Items in List 2: 3 1 5

Sample Output

Output: 1 2 3 4 5 9 10

Sample Input

Size of the list 1: 3

Items in List 1: 90 60 40

Size of List 2: 2

Items in List 2: 10 20

Sample Output

Output: 10 20 40 60 90

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

// Node structure to represent each element of the linked list
struct Node {
    int data;
    Node* next;

    Node(int val) : data(val), next(nullptr) {}
};
```

```
// Function to insert a new node at the end of the linked list
void insert(Node*& head, int val) {
    Node* newNode = new Node(val);
    if (!head) {
        head = newNode;
    } else {
        Node* temp = head;
        while (temp->next) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
}

// Function to merge two sorted linked lists into another linked list in sorted order
Node* mergeSortedLists(Node* list1, Node* list2) {
    Node* mergedList = nullptr;
    Node* temp1 = list1;
    Node* temp2 = list2;

    while (temp1 && temp2) {
        if (temp1->data <= temp2->data) {
            insert(mergedList, temp1->data);
            temp1 = temp1->next;
        } else {
            insert(mergedList, temp2->data);
            temp2 = temp2->next;
        }
    }
    while (temp1) {
        insert(mergedList, temp1->data);
        temp1 = temp1->next;
    }

    while (temp2) {
        insert(mergedList, temp2->data);
        temp2 = temp2->next;
    }

    return mergedList;
}

// Function to display the elements of a linked list
void displayList(Node* head) {
    Node* temp = head;
    while (temp) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}

int main() {
    Node* list1 = nullptr;
    Node* list2 = nullptr;
    int size1, size2;
```

```
// Input for the first linked list
cout << "Size of the list 1: ";
cin >> size1;
cout << "Items in List 1: ";
for (int i = 0; i < size1; i++) {
    int val;
    cin >> val;
    insert(list1, val);
}

// Input for the second linked list
cout << "Size of the list 2: ";
cin >> size2;
cout << "Items in List 2: ";
for (int i = 0; i < size2; i++) {
    int val;
    cin >> val;
    insert(list2, val);
}

// Merge the two sorted linked lists and display the result
Node* mergedList = mergeSortedLists(list1, list2);
cout << "Output: ";
displayList(mergedList);

// Free the memory occupied by the linked lists
while (list1) {
    Node* temp = list1;
    list1 = list1->next;
```

```
        delete temp;
    }
    while (list2) {
        Node* temp = list2;
        list2 = list2->next;
        delete temp;
    }
    while (mergedList) {
        Node* temp = mergedList;
        mergedList = mergedList->next;
        delete temp;
    }

    return 0;
}
```

Evaluation - 4, Course: Data Structures and Algorithms-I Sessional

1. Declare and initialize the necessary variables and data structures:

○ An array to store vertex values.

○ An array to track the color of each vertex during BFS.

○ A 2D array representing the adjacency matrix of the graph.

○ Initialize all elements of the arrays.

2. Define the graph connections:

○ Set specific edges in g[x][y] to 1, indicating the presence of connections between x and y vertices.

3. Initialize a queue (q) and enqueue the starting vertex:

○ Create a queue (q) to store vertices for BFS traversal.

○ Enqueue the starting vertex (source vertex) into the queue.

○ Set the color of the vertex to grey, indicating that vertex 1 is visited.

4. Perform BFS traversal:

○ While the queue is not empty, repeat the following steps:

■ Dequeue a vertex from the front of the queue and assign it to a variable.

■ Print the value of the dequeued element, indicating that the vertex has been visited.

■ If the color of a vertex (let's say k) is grey (indicating the vertex is visited but not processed):

■ Find the adjacent vertices of the vertex k.

■ Enqueue the adjacent vertices of k into the queue.

■ Set the color of the adjacent vertices of k to grey.

■ As all the adjacent vertices of k are visited, set the color of k to black

5. Continue the BFS traversal until the queue becomes empty.

6. Terminate the program.

SOLN:

```
#include<bits/stdc++.h>
using namespace std;
int count_=0;
int g[5][5]={0};
int v[5];
int color[5]={0};
int dist[5]={0};
void dfs_visit(int k)
{
cout<<k<< " "<<dist[k]<<endl;
color[k]=1;
for(int i=0;i<5;i++)
{
if(g[k][i]==1 && color[i]==0)
{
dist[i]=dist[k]+1;
dfs_visit(i);
}
}
color[k]=2;
}
void parent()
{
for(int i=0; i<5; i++)
for(int j=0; j<5; j++)
{
if(g[i][j]==1)
cout<<j<<"->"<<i<<endl;
}
}
int main()
{
g[0][1]=1;
g[0][2]=1;
g[1][4]=1;
g[1][3]=1;
dist[0]=0;
dfs_visit(0);
parent();
}
```

Evaluation on BST, Course: CSE 204: Data Structure & Algorithm I Sessional

Create a program to manage the inventory of a bookstore. The bookstore has a large collection of books, and you need to implement a binary search tree (BST) to store and retrieve book information efficiently.

Each book has the following attributes:

● ISBN (a unique identifier for the book)

● Title

● Author

● Price

● Quantity (number of copies available in the inventory)

Your program should support the following operations:

Add a book to the inventory: Given the book's details, add it to the BST.

Remove a book from the inventory: Given the ISBN, remove the book from the BST.

Update the quantity of a book: Given the ISBN and a new quantity, update the book's quantity in the BST.

Search for a book by ISBN: Given the ISBN, retrieve and display the book's details.

Display all books in the inventory: Traverse the BST and display the details of all books in the inventory.

Sorted List of the ISBN : Traverse the BST and print ISBN numbers in sorted order using DFS.

Your implementation should ensure the BST remains balanced to search, insert, and delete books. The books should be sorted in the BST based on their ISBN.

Note: You can assume that the ISBNs are unique and don't need to handle duplicate ISBNs in the inventory.

Sample Input - Output

```
Welcome to the Bookstore Inventory Management System!
1. Add a book to the inventory
2. Remove a book from the inventory
3. Update the quantity of a book
```

```
4. Search for a book by ISBN
5. Display all books in the inventory
6. Print ISBN numbers of books in sorted order
7. Exit
Enter your choice: 1
Enter the book's ISBN: 101
Enter the book's title: Clean Code
Enter the book's author: Robert C. Martin
Enter the book's price: 39.99
Enter the book's quantity: 10
Book added to the inventory successfully.
Enter your choice: 1
Enter the book's ISBN: 100
Enter the book's title: Design Patterns
Enter the book's author: Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides
Enter the book's price: 49.99
Enter the book's quantity: 5
Book added to the inventory successfully.
Enter your choice: 1
Enter the book's ISBN: 103
Enter the book's title: Data Communications and Networking
Enter the book's author: Behrouz A. Forouzan
Enter the book's price: 50.55
Enter the book's quantity: 8
Book added to the inventory successfully.

Enter your choice: 4
Enter the ISBN of the book to search: 100
ISBN: 100
```

```
Title: Design Patterns

Author: Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides

Price: $49.99

Quantity: 5


Enter your choice: 5

Inventory:

ISBN: 100

Title: Design Patterns

Author: Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides

Price: $49.99

Quantity: 5

ISBN: 101

Title: Clean Code

Author: Robert C. Martin

Price: $39.99

Quantity: 10

ISBN: 103

Title: Data Communications and Networking

Author: Behrouz A. Forouzan

Price: $50.55

Quantity: 8

Enter your choice: 2

Enter the ISBN of the book to remove: 101

Book removed from the inventory successfully.

Enter your choice: 6

Inventory:

Sorted List of ISBN : 100 103
```