DATA HOME TASK

LAB #8

Name: Ayaan Amer

Roll No: 24F-0767

Section: 3E

Q1:

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

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

class Queue {
    Node* front;
    Node* rear;

public:
    Queue() : front(nullptr), rear(nullptr) {}

    bool isEmpty() {
        return front == nullptr;
```

```
}

void enqueue(int val) {
   Node* temp = new Node(val);
   if (rear == nullptr) {
      front = rear = temp;
   }
   else {
      rear->next = temp;
      rear = temp;
   }
}

int dequeue() {
   if (isEmpty()) return -1;
   Node* temp = front;
   int val = temp->data;
   front = front->next;
   if (front == nullptr) rear = nullptr;
   delete temp;
   return val;
}

Node* getFront() {
   return front;
```

```
    }
};

int countNodes(Node* head) {
    int count = 0;
    while (head) {
        count++;
        head = head->next;
    }
    return count;
}

void duplicate(Node*& head) {
    int n = countNodes(head);
    Queue q;
    Node* temp = head;
    while (temp) {
        q.enqueue(temp->data);
        temp = temp->next;
    }

    Node* Nhead = nullptr;
    Node* tail = nullptr;

    while (!q.isEmpty()) {
```

```
            int val = q.dequeue();
            for (int i = 0; i < n + 1; i++) {
                Node* New = new Node(val);
                if (Nhead == nullptr) {
                    Nhead = tail = New;
                }
                else {
                    tail->next = New;
                    tail = New;
                }
            }
        }

        head = Nhead;
    }

    void display(Node* head) {
        while (head) {
            cout << head->data;
            if (head->next)
                cout << "->";
            head = head->next;
        }
        cout << endl;
    }
```

```
int main() {
    Queue q;
    q.enqueue(3);
    q.enqueue(4);
    q.enqueue(5);

    Node* head = q.getFront();

    cout << "Original List: ";
    display(head);

    duplicate(head);

    cout << "Duplicated List: ";
    display(head);

    return 0;
}
```

```
Original List: 3->4->5
Duplicated List: 3->3->3->3->4->4->4->4->5->5->5->5
```

Q2:

```
#include <iostream>
#include <string>
```

```
using namespace std;

struct Task {
   int id;
   string description;
   string status;
   Task* next;
   Task(int i, const string& desc) : id(i), description(desc), status("Pending"), next(nullptr)
{}
};

class Queue {
   Task* front;
   Task* rear;

public:
   Queue() : front(nullptr), rear(nullptr) {}

   bool isEmpty() { return front == nullptr; }

   void enqueue(Task* task) {
      if (rear == nullptr) {
         front = rear = task;
      }
      else {
         rear->next = task;
         rear = task;
      }
```

```
        task->next = nullptr;
    }

    Task* dequeue() {
        if (isEmpty()) return nullptr;
        Task* temp = front;
        front = front->next;
        if (front == nullptr) rear = nullptr;
        return temp;
    }

    Task* peek() { return front; }

    void display() {
        if (isEmpty()) {
            cout << "No pending tasks." << endl;
            return;
        }
        Task* curr = front;
        while (curr) {
            cout << "ID: " << curr->id
                 << ", Description: " << curr->description
                 << ", Status: " << curr->status << endl;
            curr = curr->next;
        }
    }
};
```

```
class Stack {
    Task* top;

public:
    Stack() : top(nullptr) {}

    bool isEmpty() { return top == nullptr; }

    void push(Task* task) {
        task->next = top;
        top = task;
    }

    Task* pop() {
        if (isEmpty()) return nullptr;
        Task* temp = top;
        top = top->next;
        return temp;
    }

    Task* peek() { return top; }

    void display() {
        if (isEmpty()) {
            cout << "No paused tasks." << endl;
            return;
        }
        Task* curr = top;
```

```
        while (curr) {
            cout << "ID: " << curr->id
                 << ", Description: " << curr->description
                 << ", Status: " << curr->status << endl;
            curr = curr->next;
        }
    }
};

class Scheduler {
    Queue taskQueue;
    Stack pausedStack;
    int nextId = 1;

public:
    void addTask(const string& description) {
        Task* task = new Task(nextId++, description);
        taskQueue.enqueue(task);
        cout << "Task added." << endl;
    }

    void executeTask() {
        Task* task = taskQueue.dequeue();
        if (!task) {
            cout << "No tasks to execute." << endl;
            return;
        }
        task->status = "Completed";
```

```
    cout << "Executed Task ID: " << task->id
         << ", Description: " << task->description << endl;
    delete task;
}

void pauseTask() {
    Task* task = taskQueue.dequeue();
    if (!task) {
        cout << "No task to pause." << endl;
        return;
    }
    task->status = "Paused";
    pausedStack.push(task);
    cout << "Task paused." << endl;
}

void resumeTask() {
    Task* task = pausedStack.pop();
    if (!task) {
        cout << "No paused task to resume." << endl;
        return;
    }
    task->status = "Pending";
    taskQueue.enqueue(task);
    cout << "Task resumed." << endl;
}

void pending() {
```

```
        cout << endl << "Pending Tasks:" << endl;
        taskQueue.display();
    }

    void viewPausedTasks() {
        cout << endl << "Paused Tasks:" << endl;
        pausedStack.display();
    }
};

int main() {
    Scheduler scheduler;
    int choice;
    string desc;

    while (true) {
            cout << "1. Add Task" << endl;
            cout << "2. Execute Task" << endl;
            cout << "3. Pause Task" << endl;
            cout << "4. Resume Task" << endl;
            cout << "5. View Pending Tasks" << endl;
            cout << "6. View Paused Tasks" << endl;
            cout << "7. Exit" << endl;
            cout << "Enter choice ";
        cin >> choice;
        cin.ignore();

        if (choice == 1) {
```

```
        cout << "Enter task description ";
        getline(cin, desc);
        scheduler.addTask(desc);
    }
    else if (choice == 2) {
        scheduler.executeTask();
    }
    else if (choice == 3) {
        scheduler.pauseTask();
    }
    else if (choice == 4) {
        scheduler.resumeTask();
    }
    else if (choice == 5) {
        scheduler.pending();
    }
    else if (choice == 6) {
        scheduler.viewPausedTasks();
    }
    else if (choice == 7) {
        cout << "Exiting." << endl;
        break;
    }
    else {
        cout << "Invalid choice" << endl;
    }
}
```

```
    return 0;
}
```

```
1. Add Task
2. Execute Task
3. Pause Task
4. Resume Task
5. View Pending Tasks
6. View Paused Tasks
7. Exit
Enter choice 1
Enter task description qwer
Task added.
1. Add Task
2. Execute Task
3. Pause Task
4. Resume Task
5. View Pending Tasks
6. View Paused Tasks
7. Exit
Enter choice 3
Task paused.
1. Add Task
2. Execute Task
3. Pause Task
4. Resume Task
5. View Pending Tasks
6. View Paused Tasks
7. Exit
Enter choice 4
Task resumed.
1. Add Task
2. Execute Task
```

Q3:

```
#include <iostream>

using namespace std;


struct Node {
    int data;
```

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

Node* merge(Node* l1, Node* l2) {
    if (!l1) return l2;
    if (!l2) return l1;

    if (l1->data < l2->data) {
        l1->next = merge(l1->next, l2);
        return l1;
    }
    else {
        l2->next = merge(l1, l2->next);
        return l2;
    }
}
void insert(Node*& head, int val) {
    if (!head) {
        head = new Node(val);
        return;
    }
    Node* temp = head;
    while (temp->next) temp = temp->next;
    temp->next = new Node(val);
}
void print(Node* head) {
    while (head) {
```

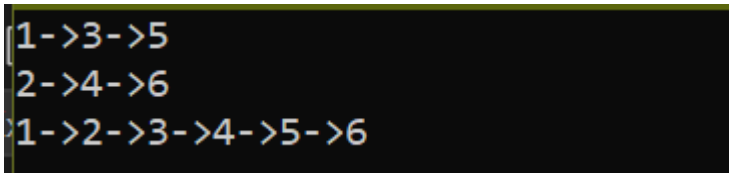
```
        cout << head->data;
        if (head->next) cout << "->";
        head = head->next;
    }
    cout << endl;
}
int main() {
    Node* list1 = nullptr;
    Node* list2 = nullptr;
    insert(list1, 1);
    insert(list1, 3);
    insert(list1, 5);
    insert(list2, 2);
    insert(list2, 4);
    insert(list2, 6);

    print(list1);
    print(list2);

    Node* merged = merge(list1, list2);

    print(merged);

    return 0;
}
```



Q3:

```
#include <iostream>

using namespace std;


struct Node {

   int data;

   Node* next;

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

};


Node* merge(Node* l1, Node* l2) {

   if (!l1) return l2;

   if (!l2) return l1;


   if (l1->data < l2->data) {

      l1->next = merge(l1->next, l2);

      return l1;

   }

   else {

      l2->next = merge(l1, l2->next);

      return l2;

   }

}

void insert(Node*& head, int val) {

   if (!head) {
```

```
        head = new Node(val);
        return;
    }
    Node* temp = head;
    while (temp->next) temp = temp->next;
    temp->next = new Node(val);
}
void print(Node* head) {
    while (head) {
        cout << head->data;
        if (head->next) cout << "->";
        head = head->next;
    }
    cout << endl;
}
int main() {
    Node* list1 = nullptr;
    Node* list2 = nullptr;
    insert(list1, 1);
    insert(list1, 3);
    insert(list1, 5);
    insert(list2, 2);
    insert(list2, 4);
    insert(list2, 6);

    print(list1);
    print(list2);
```

```
    Node* merged = merge(list1, list2);

    print(merged);

    return 0;
}
```

Q4:

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

struct Person {
    string name;
    string cnic;
    int age;
    Person* next;
    Person(string n, string c, int a) : name(n), cnic(c), age(a), next(nullptr) {}
};

class PriorityQueue {
    Person* front;
    Person* rear;

public:
    PriorityQueue() : front(nullptr), rear(nullptr) {}

    bool isEmpty() {
        return front == nullptr;
```

```
}

void enqueue(string name, string cnic, int age) {
    Person* newPerson = new Person(name, cnic, age);

    if (!front) {
        front = rear = newPerson;
    }
    else if (age > front->age) {
        newPerson->next = front;
        front = newPerson;
    }
    else if (age <= rear->age) {
        rear->next = newPerson;
        rear = newPerson;
    }
    else {
        Person* temp = front;
        while (temp->next && temp->next->age >= age) {
            temp = temp->next;
        }
        newPerson->next = temp->next;
        temp->next = newPerson;
    }
}

void dequeue() {
    if (isEmpty()) {
```

```
            cout << "No person in queue." << endl;
            return;
        }
        Person* temp = front;
        cout << temp->name << " please come for bill payment." << endl;
        front = front->next;
        if (!front) rear = nullptr;
        delete temp;
    }

    void display() {
        if (isEmpty()) {
            cout << "Queue is empty." << endl;
            return;
        }
        Person* temp = front;
        while (temp) {
            cout << "Name: " << temp->name << ", CNIC: " << temp->cnic << ", Age: " << temp->age << endl;
            temp = temp->next;
        }
    }
};

int main() {
    PriorityQueue pq;
    int choice;
    string name, cnic;
```

```
int age;

while (true) {
   cout << "Press 1 to enter information" << endl;
   cout << "Press 2 to call for bill Payment" << endl;
   cout << "Press 3 to view all persons" << endl;
   cout << "Press 4 to Exit" << endl;
   cout << "Enter choice: ";
   cin >> choice;
   cin.ignore();

   if (choice == 1) {
      cout << "Enter your Name: ";
      getline(cin, name);
      cout << "Enter your CNIC: ";
      getline(cin, cnic);
      cout << "Enter your Age: ";
      cin >> age;
      cin.ignore();
      pq.enqueue(name, cnic, age);
   }
   else if (choice == 2) {
      pq.dequeue();
   }
   else if (choice == 3) {
      pq.display();
   }
   else if (choice == 4) {
```

```
            cout << "Exiting" << endl;
            break;
        }
        else {
            cout << "Invalid choice" << endl;
        }
    }

    return 0;
}
```

```
Enter choice: 1
Enter your Name: rmt
Enter your CNIC: 23456
Enter your Age: 19
Press 1 to enter information
Press 2 to call for bill Payment
Press 3 to view all persons
Press 4 to Exit
Enter choice: 1
Enter your Name: RM
Enter your CNIC: 2345
Enter your Age: 24
Press 1 to enter information
Press 2 to call for bill Payment
Press 3 to view all persons
Press 4 to Exit
Enter choice: 2
RM please come for bill payment.
Press 1 to enter information
Press 2 to call for bill Payment
Press 3 to view all persons
Press 4 to Exit
Enter choice: 3
Name: rmt, CNIC: 23456, Age: 19
Press 1 to enter information
Press 2 to call for bill Payment
Press 3 to view all persons
Press 4 to Exit
```