**Question 1:**

***Print Linked List in Reverse Order (Iteration);***

*// prints in reverse (iteration)*

*void* display\_reverse(){

    student \*k = back;

    student \*p = front;

    student \*p\_2;

    if(p == k){

        cout << "Student ID is: " << p -> id;

        cout << "\tStudent Marks are: " << p -> marks << endl;

    }

    else{

        while(true){

            while(p != k){

                p\_2 = p;

                p = p -> next;

            }

            cout << "Student ID is: " << p -> id;

            cout << "\tStudent Marks are: " << p -> marks << endl;

            k = p\_2;

            p = front;

            if(k == front){

                cout << "Student ID is: " << p -> id;

                cout << "\tStudent Marks are: " << p -> marks << endl;

                break;

            }

        }

    }

***Print Linked List in Reverse Order (Recursion);***

*// prints in reverse using recursion*

*void* print\_backward\_recurrsion(student *\*x*){

    if(*x* -> next != NULL){

        print\_backward\_recurrsion(*x* -> next);

        cout << "Student ID is: " << *x* -> id;

        cout << "\tStudent marks is: " << *x* -> marks << endl;

    }

    else{

        cout << "Student ID is: " << *x* -> id;

        cout << "\tStudent marks is: " << *x* -> marks << endl;

*x* = NULL;

    }

}

**Question 2:**

***Print first -> last, second -> second last and so on***

*// prints frist and last and so on*

*void* front\_and\_back\_display(){

    student \*k = back;

    student \*p = front;

    student \*p\_2;

    student \*p\_3 = front;

    cout << "\n----------------- DISPLAY ------------------\n" << endl;

    while(true){

        while(p != k){

            p\_2 = p;

            p = p -> next;

        }

        cout << "Student ID is: " << p\_3 -> id;

        cout << "\tStudent Marks are: " << p\_3 -> marks << endl;

        p\_3 = p\_3 -> next;

        cout << "Student ID is: " << k -> id;

        cout << "\tStudent Marks are: " << k -> marks << endl;

*// odd check*

        if(p\_3 -> next == k){

            cout << "Student ID is: " << p\_3 -> id;

            cout << "\tStudent Marks are: " << p\_3 -> marks << endl;

            break;

        }

        k = p\_2;

        p = front;

*// even check*

        if(p\_3 -> next == k){

            cout << "Student ID is: " << p\_3 -> id;

            cout << "\tStudent Marks are: " << p\_3 -> marks << endl;

            cout << "Student ID is: " << k -> id;

            cout << "\tStudent Marks are: " << k -> marks << endl;

            break;

        }

    }

}

**Question 3:**

***Palindrome Check;***

*// checks if linked list is palindrome on the basis of ID*

*void* palindrome\_check(){

student \*k = back;

student \*p = front;

student \*p\_2;

student \*p\_3 = front;

*bool* check = true;

    while(true){

        while(p != k){

            p\_2 = p;

            p = p -> next;

        }

        if((p\_3 -> id != k -> id)){

            check = false;

            break;

        }

*// k getting updated*

        k = p\_2;

        p = front;

*// p\_3 getting updated*

        p\_3 = p\_3 -> next;

        if(p\_3 -> next == NULL){

            break;

        }

    }

    if(check == true){

        cout << "Palindrome" << endl;

    }

    else{

        cout << "Not Palindrome" << endl;

    }

}

**Question 4:**

***Swap ID’s of two nodes***

*// ID based swap*

*void* swapID(*int* *value1*, *int* *value2*){

*// pointers to locate nodes*

    student \*p = front;

    student \*p2 = front;

    while (p -> id != *value1* && p != NULL)

    {

        p = p -> next;

    }

    while (p2 -> id != *value2* && p2 != NULL)

    {

        p2 = p2 -> next;

    }

*// swaping and displaying*

*int* temp = p -> id;

    p -> id = p2 -> id;

    p2 -> id = temp;

    display();

}

**Question 5:**

***Careem Linked List!***

*struct* rides{

*int* ride\_id;

*int* ride\_distance;

*int* ride\_charges;

    string rider\_name;

*// Pointer for putting more values*

    rides \*next = NULL;

};

*struct* registered\_cars{

*int* reg\_number;

*int* car\_type;

    string driver\_name;

    string car\_name;

*// car pointers*

    registered\_cars \*previous = NULL;

    registered\_cars \*next = NULL;

    rides \*s\_next = NULL;

};

registered\_cars \*front = NULL;

registered\_cars \*back = NULL;

*// Student Methods*

*void* front\_id(){

    cout << "ID at front is: " << front -> reg\_number << endl;

}

*void* back\_id(){

    cout << "ID at back is: " << back -> reg\_number << endl;

}

*bool* isEmpty(){

    if(front == NULL){

        return true;

    }

    else{

        return false;

    }

}

*void* insert\_at\_end(){

    registered\_cars \*current = new registered\_cars;

    cout << "\nEnter Registration Number " << endl;

    cin >> current -> reg\_number;

    cout << "\nEnter Driver Name" << endl;

    cin >> current -> driver\_name;

    cout << "\nEnter Car Name" << endl;

    cin >> current -> car\_name;

    cout << "\nEnter Car Type" << endl;

    cin >> current -> car\_type;

    if(isEmpty()){

        front = back = current;

    }

    else{

        current -> previous = back;

        back -> next = current;

        back = current;

    }

}

*void* delete\_specfic(*int* *value*){

    registered\_cars \*p = front;

    if(isEmpty()){

        cout << "Empty!" << endl;

    }

    else if(*value* == front -> reg\_number){

        p = front -> next;

        p -> previous = NULL;

        delete front;

        front = p;

    }

    else if(*value* == back -> reg\_number){

        p = back -> previous;

        p -> next = NULL;

        delete back;

        back = p;

    }

    else{

        while(p -> reg\_number != *value* && p != NULL){

            p = p -> next;

        }

        p -> next -> previous = p -> previous;

        p -> previous -> next = p -> next;

        delete p;

    }

}

*void* display(){

    registered\_cars \*p = front;

    if(isEmpty()){

        cout << "No record!" << endl;

    }

    else{

        while (p != NULL)

        {

            cout << "\n----------------- DISPLAY ------------------\n\nRegistration Number: " << p -> reg\_number << endl;

            cout << "\nDriver Name:         " << p -> driver\_name << endl;

            cout << "\nCar Name:            " << p -> car\_name << endl;

            cout << "\nCar Type:            " << p -> car\_type << endl;

            p = p -> next;

        }

    }

}

*void* display\_cars\_type(*int* *value*){

    registered\_cars \*p = front;

    cout << "\n----------------- DISPLAY ------------------ " << endl;

    while (p != NULL){

        if(p -> car\_type == 2){

            cout << "Type 2 car is: " << p -> car\_name << endl;

        }

        p = p -> next;

    }

}

*void* insert\_rides(*int* *value*){

    registered\_cars \*p = front;

    rides \*p3;

    rides \*ride = new rides();

    cout << "\nEnter Ride ID" << endl;

    cin >> ride -> ride\_id;

    cout << "\nEnter Ride Distance" << endl;

    cin >> ride -> ride\_distance;

    cout << "\nEnter Ride Charges" << endl;

    cin >> ride -> ride\_charges;

    cout << "\nEnter Rider Name" << endl;

    cin >> ride -> rider\_name;

    while(p -> reg\_number != *value* && p != NULL){

        p = p -> next;

    }

    if(p -> s\_next == NULL){

        p -> s\_next = ride;

    }

    else{

        rides \*p2 = p -> s\_next;

        while(p2 != NULL){

            p3 = p2;

            p2 = p2 -> next;

        }

        cout << "reached";

        p3 -> next = ride;

    }

}

*// Displays all rides info*

*void* display\_all\_rides(*int* *value*){

    registered\_cars \*p = front;

    while(p -> reg\_number != *value* && p != NULL){

        p = p -> next;

    }

    if(p -> s\_next == NULL){

        cout << "No record!" << endl;

    }

    else{

        rides \*p2 = p -> s\_next;

        while (p2 != NULL)

        {

            cout << "\n----------------- DISPLAY ------------------\n\nRide ID: " << p2 -> ride\_id << endl;

            cout << "\nRide Distance: " << p2 -> ride\_distance << endl;

            cout << "\nRide Charges: " << p2 -> ride\_charges << endl;

            cout << "\nRider Name: " << p2 -> rider\_name << endl;

*// Moving forward*

            p2 = p2 -> next;

        }

    }

}

*// Displays total earning of a specfic car*

*void* calculate\_specfic\_earning(*int* *value*){

    registered\_cars \*p = front;

*int* total\_earning = 0;

    while(p -> reg\_number != *value* && p != NULL){

        p = p -> next;

    }

    if(p -> s\_next == NULL){

        cout << "No record!" << endl;

    }

    else{

        rides \*p2 = p -> s\_next;

        while (p2 != NULL)

        {

            total\_earning += p2 -> ride\_charges;

*// Moving forward*

            p2 = p2 -> next;

        }

    }

    cout << "\n----------------- DISPLAY ------------------\n" << endl;

    cout << "Total earnings of this car is: " << total\_earning << endl;

}

*// Displays total earning of all cars*

*void* calculate\_total\_earning(){

    registered\_cars \*p = front;

*int* total\_earning\_singleCar = 0;

*int* total\_earnings = 0;

    while(p != NULL){

        rides \*p2 = p -> s\_next;

        if(p -> s\_next == NULL){

            cout << "No record!" << endl;

        }

        else{

            while (p2 != NULL)

            {

                total\_earning\_singleCar += p2 -> ride\_charges;

*// Moving forward*

                p2 = p2 -> next;

            }

        }

        total\_earnings += total\_earning\_singleCar;

        total\_earning\_singleCar = 0;

        p = p -> next;

    }

    cout << "\n----------------- DISPLAY ------------------\n" << endl;

    cout << "Total earnings of all the cars is: " << total\_earnings << endl;

}

**Question 6:**

***Find max of two list and insert into another using circular linked list!***

*struct* circular{

*int* number1;

*int* number2;

*int* number3;

    circular \*next;

};

circular \*back\_List1 = NULL;

circular \*back\_List2 = NULL;

circular \*back\_ListMax = NULL;

*void* front\_id(){

    cout << "ID at front is: " << back\_List1 -> next -> number1 << endl;

}

*void* back\_id(){

    cout << "ID at back is: " << back\_List1 -> number1 << endl;

}

*void* front\_id\_2(){

    cout << "ID at front is: " << back\_List2 -> next -> number2 << endl;

}

*void* back\_id\_2(){

    cout << "ID at back is: " << back\_List2 -> number2 << endl;

}

*int* length(){

    circular \*p = back\_List1;

*int* count = 0;

    do

    {

        p = p -> next;

        count++;

    } while (p != back\_List1);

    return count;

}

*bool* isEmpty\_List1(){

    if(back\_List1 == NULL){

        return true;

    }

    else{

        return false;

    }

}

*bool* isEmpty\_List2(){

    if(back\_List2 == NULL){

        return true;

    }

    else{

        return false;

    }

}

*bool* isEmpty\_ListMax(){

    if(back\_ListMax == NULL){

        return true;

    }

    else{

        return false;

    }

}

*void* insert\_at\_end\_List1(){

    circular \*current = new circular;

    cout << "\nEnter number" << endl;

    cin >> current -> number1;

    if(isEmpty\_List1()){

        back\_List1 = current;

        back\_List1 -> next = current;

    }

    else{

        current -> next = back\_List1 -> next;

        back\_List1 -> next = current;

        back\_List1 = current;

    }

}

*void* insert\_at\_end\_List2(){

    circular \*current = new circular;

    cout << "\nEnter number" << endl;

    cin >> current -> number2;

    if(isEmpty\_List2()){

        back\_List2 = current;

        back\_List2 -> next = current;

    }

    else{

        current -> next = back\_List2 -> next;

        back\_List2 -> next = current;

        back\_List2 = current;

    }

}

*// Main method that does all the job!*

*void* set\_max\_values(){

*// essentials for finding max in list one!*

    circular \*p1 = back\_List1;

    circular \*p2; *// Previous Pointer*

    circular \*value\_at\_max\_1 = NULL;

    circular \*previous\_at\_max\_1 = NULL;

*int* max\_1 = 0; *// Max value of circular list 1*

*// essentials for finding max in list two!*

    circular \*p3 = back\_List2;

    circular \*p4; *// Previous Pointer*

    circular \*value\_at\_max\_2 = NULL;

    circular \*previous\_at\_max\_2 = NULL;

*int* max\_2 = 0; *// Max value of circular list 2*

    do

    {

        do

        {

            p2 = p1; *// Pointing previous number*

            p1 = p1 -> next;

            if(p1 -> number1 >= max\_1){

                max\_1 = p1 -> number1;

                value\_at\_max\_1 = p1;

                previous\_at\_max\_1 = p2;

            }

        } while (p1 != back\_List1);

        cout << "Max of List 1 is: " << max\_1 << endl;

        circular \*current = new circular();

*//current = value\_at\_max\_1;*

        current -> number3 = max\_1;

        if(isEmpty\_ListMax()){

            back\_ListMax = current;

            current -> next = back\_ListMax;

        }

        else{

            current -> next = back\_ListMax -> next;

            back\_ListMax -> next = current;

            back\_ListMax = current;

        }

*// Checks the Only element left in the list*

        if(p1 -> next == p1){

*//cout << "Only number in the list is: " << p1 -> number1 << endl;*

            back\_List1 = NULL;

            p1 = NULL;

        }

*// Checks the first element*

        else if(value\_at\_max\_1 -> number1 == back\_List1 -> next -> number1){

            cout << "Reached First" << endl;

            back\_List1 -> next = back\_List1 -> next -> next;

        }

*// Checks the last element*

        else if(value\_at\_max\_1 -> number1 == back\_List1 -> number1){

            cout << "Reached last" << endl;

            previous\_at\_max\_1 -> next = back\_List1 -> next;

            back\_List1 = previous\_at\_max\_1;

        }

*// Checks the alternate element*

        else{

            cout << "Reached alternate" << endl;

            previous\_at\_max\_1 -> next = value\_at\_max\_1 -> next;

        }

        max\_1 = 0;

        do

        {

            p4 = p3; *// Pointing previous number*

            p3 = p3 -> next;

            if(p3 -> number2 >= max\_2){

                max\_2 = p3 -> number2;

                value\_at\_max\_2 = p3;

                previous\_at\_max\_2 = p4;

            }

        } while (p3 != back\_List2);

        cout << "Max of List 2 is: " << max\_2 << endl;

        circular \*current2 = new circular();

        current2 -> number3 = max\_2;

        current2 -> next = back\_ListMax -> next;

        back\_ListMax -> next = current2;

        back\_ListMax = current2;

*// Checks the Only element left in the list*

        if(p3 -> next == p3){

*// cout << "Only number in the list is: " << p3 -> number2 << endl;*

            back\_List2 = NULL;

            p3 = NULL;

        }

*// Checks the first element*

        else if(value\_at\_max\_2 -> number2 == back\_List2 -> next -> number2){

            cout << "Reached First" << endl;

            back\_List2 -> next = back\_List2 -> next -> next;

        }

*// Checks the last element*

        else if(value\_at\_max\_2 -> number2 == back\_List2 -> number2){

            cout << "Reached last" << endl;

            previous\_at\_max\_2 -> next = back\_List2 -> next;

            back\_List2 = previous\_at\_max\_2;

        }

*// Checks the alternate element*

        else{

            cout << "Reached alternate" << endl;

            previous\_at\_max\_2 -> next = value\_at\_max\_2 -> next;

        }

        max\_2 = 0;

    } while(p1 != NULL && p3 != NULL);

}