**K230607 DS LAB # 3 Sept 7,2024**

**Singly Linked List**

**Quest # 1**

#include <iostream>

using namespace std;

class Node

{

public:

    int val;

    Node \*next;

    Node(int data)

    {

        val = data;

        next = nullptr;

    }

};

void InsertHead(Node \*&head, int data)

{

    Node \*temp = new Node(data);

    temp->next = head;

    head = temp;

}

void InsertTail(Node \*&head, int data)

{

    Node \*temp = new Node(data);

    if (!head)

    {

        head = temp;

    }

    else

    {

        Node \*ptr = head;

        while (ptr->next != nullptr)

        {

            ptr = ptr->next;

        }

        ptr->next = temp;

    }

}

void reverse(Node \*&head)

{

    Node \*forward = nullptr;

    Node \*start = head;

    Node \*prev = nullptr;

    while (start)

    {

        forward = start->next;

        start->next = prev;

        prev = start;

        start = forward;

    }

    head = prev;

}

void Insert(Node \*&head, Node \*&ans)

{

    if (!head)

    {

        cout << "Invalid" << endl;

    }

    else

    {

        Node \*ptr = head;

        Node \*even = nullptr;

        Node \*odd = nullptr;

        while (ptr != nullptr)

        {

            if (ptr->val % 2 == 0)

            {

                InsertHead(even, ptr->val);

            }

            else

            {

                InsertHead(odd, ptr->val);

            }

            ptr = ptr->next;

        }

        if (!even)

        {

            ans = odd;

        }

        else if (!odd)

        {

            ans = even;

        }

        else

        {

            ans = even;

            Node \*res = ans;

            while (res->next)

            {

                res = res->next;

            }

            res->next = odd;

        }

    }

}

void print(Node \*head)

{

    while (head)

    {

        cout << head->val << " ";

        head = head->next;

    }

    cout << endl;

}

int main()

{

    Node \*ans = nullptr;

    Node \*head = new Node(17);

    head->next = new Node(15);

    head->next->next = new Node(8);

    head->next->next->next = new Node(12);

    head->next->next->next->next = new Node(10);

    head->next->next->next->next->next = new Node(5);

    head->next->next->next->next->next->next = new Node(4);

    head->next->next->next->next->next->next->next = new Node(1);

    head->next->next->next->next->next->next->next->next = new Node(7);

    head->next->next->next->next->next->next->next->next->next = new Node(6);

    cout << "Original : ";

    print(head);

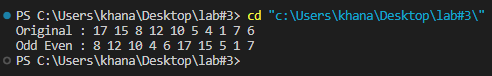
    reverse(head);

    Insert(head, ans);

    cout << "Odd Even : ";

    print(ans);

}

****

**Quest # 2**

#include <iostream>

using namespace std;

class Node

{

public:

    int val;

    Node \*next;

    Node(int data)

    {

        val = data;

        next = nullptr;

    }

};

bool isPalindrome(Node \*&head)

{

    if (!head || !head->next)

    {

        return true;

    }

    Node \*slow = head;

    Node \*fast = head;

    // Middle Node

    while (fast && fast->next)

    {

        slow = slow->next;

        fast = fast->next->next;

    }

    // Reverse

    Node \*forward = nullptr;

    Node \*start = head;

    Node \*prev = nullptr;

    while (start != slow)

    {

        forward = start->next;

        start->next = prev;

        prev = start;

        start = forward;

    }

//if odd num of nodes

    if (fast)

    {

        slow = slow->next;

    }

    Node \*firstHalf = prev;

    Node \*secondHalf = slow;

    while (firstHalf && secondHalf)

    {

        if (firstHalf->val != secondHalf->val)

        {

            return false;

        }

        firstHalf = firstHalf->next;

        secondHalf = secondHalf->next;

    }

    return true;

}

void print(Node \*head)

{

    while (head)

    {

        cout << head->val << " ";

        head = head->next;

    }

    cout << endl;

}

int main()

{

    Node \*head = new Node(1);

    head->next = new Node(2);

    head->next->next = new Node(2);

    head->next->next->next = new Node(1);

    print(head);

    if (!isPalindrome(head))

    {

        cout << "Is Not Palindrome" << endl;

    }

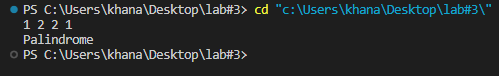
    else

    {

        cout << "Palindrome" << endl;

    }

}

****

**Quest # 3**

#include <iostream>

using namespace std;

class Node

{

public:

    int val;

    Node \*next;

    Node(int data)

    {

        val = data;

        next = nullptr;

    }

};

void InsertTail(Node \*&head, Node \*&tail, int data)

{

    Node \*temp = new Node(data);

     if (!head)

    {

        head = temp;

        tail = temp;

        tail->next = head;

    }

    else

    {

        tail->next = temp;

        tail = temp;

        tail->next = head;

    }

}

void InsertHead(Node \*&head, Node \*&tail, int data)

{

    Node \*temp = new Node(data);

    if (!head)

    {

        head = temp;

        tail = temp;

        tail->next = head;

    }

    else

    {

        temp->next = head;

        head = temp;

        tail->next = head;

    }

}

void InsertPos(Node \*&head, Node \*&tail, int data, int pos)

{

    if (pos <= 0)

    {

        cout << "Invalid" << endl;

        return;

    }

    Node \*temp = new Node(data);

    if (pos == 1)

    {

        head = temp;

        tail = temp;

        tail->next = head;

    }

    else

    {

        Node \*ptr = head;

        for (int i = 1; i < pos - 1 && ptr->next != tail; i++)

        {

            ptr = ptr->next;

        }

        temp->next = ptr->next;

        ptr->next = temp;

        if (ptr == tail)

        {

            tail = temp;

            tail->next = head;

        }

    }

}void print(Node \*head)

{

    if (!head)

        return;

    Node \*ptr = head;

    do

    {

        cout << ptr->val << " ";

        ptr = ptr->next;

    } while (ptr != head);

    cout << endl;

}

void DeletePos(Node \*&head, Node \*&tail, int pos)

{

    if (!head || pos <= 0)

    {

        cout << "Invalid " << endl;

        return;

    }

    if (pos == 1)

    {

        Node \*temp = head;

        if (head == tail)

        {

            head = nullptr;

            tail = nullptr;

        }

        else

        {

            head = head->next;

            tail->next = head;

        }

        delete temp;

    }

    else

    {

        Node \*ptr = head;

        for (int i = 1; i < pos - 1 && ptr->next != head; i++)

        {

            ptr = ptr->next;

        }

        if (ptr->next == head)

        {

            cout << "Invalid" << endl;

            return;

        }

        Node \*temp = ptr->next;

        ptr->next = temp->next;

        if (temp == tail)

        {

            tail = ptr;

            tail->next = head;

        }

        delete temp;

    }

}

int main()

{

    Node \*head = nullptr;

    Node \*tail = nullptr;

    InsertHead(head, tail, 1);

    InsertHead(head, tail, 2);

    InsertHead(head, tail, 3);

    InsertHead(head, tail, 4);

    InsertTail(head, tail, 5);

    InsertTail(head, tail, 6);

    InsertPos(head, tail, 7, 3);

    InsertPos(head, tail, 11, 4);

    InsertPos(head, tail, 9, 6);

    cout << "Original: ";

    print(head);

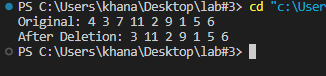
    DeletePos(head, tail, 3);

    DeletePos(head, tail, 1);

    cout << "After Deletion: ";

    print(head);

}

****

**Doubly Circular List**

**Quest # 1**

#include <iostream>

using namespace std;

class Node

{

public:

    int val;

    Node \*next;

    Node \*prev;

    Node(int data)

    {

        val = data;

        next = nullptr;

        prev = nullptr;

    }

};

void InsertTail(Node \*&head, Node \*&tail, int val)

{

    Node \*temp = new Node(val);

    if (!head)

    {

        head = temp;

        tail = temp;

        head->next = head;

        head->prev = head;

    }

    else

    {

        temp->prev = tail;

        temp->next = head;

        tail->next = temp;

        head->prev = temp;

        tail = temp;

    }

}

void InsertHead(Node \*&head, Node \*&tail, int val)

{

    Node \*temp = new Node(val);

    if (!head)

    {

        head = temp;

        tail = temp;

        head->next = head;

        head->prev = head;

    }

    else

    {

        temp->next = head;

        temp->prev = tail;

        head->prev = temp;

        tail->next = temp;

        head = temp;

    }

}

void InsertPos(Node \*&head, Node \*&tail, int pos, int val)

{

    if (pos <= 0)

    {

        cout << "Invalid " << endl;

        return;

    }

    if (pos == 1)

    {

        InsertHead(head, tail, val);

        return;

    }

    Node \*temp = head;

    int count = 1;

    while (temp && count < pos - 1)

    {

        temp = temp->next;

        count++;

        if (temp == head) break;

    }

    if (temp == head && count < pos - 1)

    {

        cout << "Invalid " << endl;

        return;

    }

    Node \*newNode = new Node(val);

    newNode->next = temp->next;

    newNode->prev = temp;

    temp->next->prev = newNode;

    temp->next = newNode;

    if (temp == tail)

    {

        tail = newNode;

    }

}

void print(Node \*head)

{

    if (!head) return;

    Node \*ptr = head;

    do

    {

        cout << ptr->val << " ";

        ptr = ptr->next;

    } while (ptr != head);

    cout << endl;

}

void printBackward(Node \*tail)

{

    if (!tail) return;

    Node \*ptr = tail;

    do

    {

        cout << ptr->val << " ";

        ptr = ptr->prev;

    } while (ptr != tail);

    cout << endl;

}

void DeletePos(Node \*&head, Node \*&tail, int pos)

{

    if (pos <= 0 || !head)

    {

        cout << "Invalid" << endl;

        return;

    }

    Node \*ptr = head;

    int count = 1;

    if (pos == 1)

    {

        if (head == tail)

        {

            delete head;

            head = nullptr;

            tail = nullptr;

        }

        else

        {

            Node \*temp = head;

            head = head->next;

            head->prev = tail;

            tail->next = head;

            delete temp;

        }

        return;

    }

    while (ptr && count < pos)

    {

        ptr = ptr->next;

        count++;

        if (ptr == head)

        {

            cout << "Invalid" << endl;

            return;

        }

    }

    if (count != pos)

    {

        cout << "Invalid" << endl;

        return;

    }

    if (ptr == tail)

    {

        tail = ptr->prev;

        tail->next = head;

        head->prev = tail;

    }

    else

    {

        ptr->prev->next = ptr->next;

        ptr->next->prev = ptr->prev;

    }

    delete ptr;

}

int main()

{

    Node\* head = nullptr;

    Node\* tail = nullptr;

    InsertTail(head, tail, 10);

    InsertHead(head, tail, 20);

    InsertPos(head, tail, 1, 15);

    cout << "Forward print : ";

    print(head);

    cout << "Backward print : ";

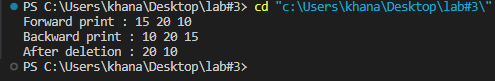
    printBackward(tail);

    DeletePos(head, tail, 1);

    cout << "After deletion : ";

    print(head);

}



**Quest # 2**

#include <iostream>

using namespace std;

class Node

{

public:

    int val;

    Node \*next;

    Node \*prev;

    Node(int data)

    {

        val = data;

        next = nullptr;

        prev = nullptr;

    }

};

void InsertTail(Node \*&head, Node \*&tail, int val)

{

    Node \*temp = new Node(val);

    if (!head)

    {

        head = temp;

        tail = temp;

        head->next = head;

        head->prev = head;

    }

    else

    {

        temp->prev = tail;

        temp->next = head;

        tail->next = temp;

        head->prev = temp;

        tail = temp;

    }

}

void Merge(Node \*head1, Node \*head2, Node \*&ans\_head, Node \*&ans\_tail)

{

    if (!head1 && !head2)

    {

        cout << "Empty Lists" << endl;

        return;

    }

    if (head1)

    {

        Node \*ptr = head1;

        do

        {

            InsertTail(ans\_head, ans\_tail, head1->val);

            head1 = head1->next;

        } while (head1 != ptr);

    }

    if (head2)

    {

        Node \*ptr = head2;

        do

        {

            InsertTail(ans\_head, ans\_tail, head2->val);

            head2 = head2->next;

        } while (head2 != ptr);

    }

}

void print(Node \*head)

{

    if (!head)

        return;

    Node \*ptr = head;

    do

    {

        cout << ptr->val << " ";

        ptr = ptr->next;

    } while (ptr != head);

    cout << endl;

}

int main()

{

    Node \*head1 = nullptr;

    Node \*tail1 = nullptr;

    Node \*head2 = nullptr;

    Node \*tail2 = nullptr;

    Node \*ans\_head = nullptr;

    Node \*ans\_tail = nullptr;

    InsertTail(head1, tail1, 1);

    InsertTail(head1, tail1, 6);

    InsertTail(head1, tail1, 8);

    InsertTail(head1, tail1, 0);

    InsertTail(head2, tail2, 3);

    InsertTail(head2, tail2, 7);

    InsertTail(head2, tail2, 9);

    cout << "List 1: ";

    print(head1);

    cout << "List 2: ";

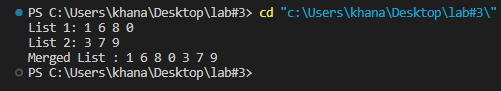
    print(head2);

    cout << "Merged List : ";

    Merge(head1, head2, ans\_head, ans\_tail);

    print(ans\_head);

}

****

**Quest # 3**

#include <iostream>

using namespace std;

class Node

{

public:

    int val;

    Node \*next;

    Node \*prev;

    Node(int data)

    {

        val = data;

        next = nullptr;

        prev = nullptr;

    }

};

void InsertTail(Node \*&head, Node \*&tail, int val)

{

    Node \*temp = new Node(val);

    if (!head)

    {

        head = temp;

        tail = temp;

        head->next = head;

        head->prev = head;

    }

    else

    {

        temp->prev = tail;

        temp->next = head;

        tail->next = temp;

        head->prev = temp;

        tail = temp;

    }

}

void swap(Node \*&ptr1, Node \*&ptr2, Node \*&head, Node \*&tail)

{

    if (ptr1 == ptr2) return;

    Node \*temp1 = ptr1->next;

    Node \*prev1 = ptr1->prev;

    Node \*temp2 = ptr2->next;

    Node \*prev2 = ptr2->prev;

    if (ptr1 == head) head = ptr2;

    else if (ptr2 == head) head = ptr1;

    if (ptr1 == tail) tail = ptr2;

    else if (ptr2 == tail) tail = ptr1;

    if (temp1 == ptr2)

    {

        ptr1->next = temp2;

        ptr1->prev = ptr2;

        ptr2->next = ptr1;

        ptr2->prev = prev1;

        if (temp2) temp2->prev = ptr1;

        if (prev1) prev1->next = ptr2;

    }

    else if (temp2 == ptr1)

    {

        ptr2->next = temp1;

        ptr2->prev = ptr1;

        ptr1->next = ptr2;

        ptr1->prev = prev2;

        if (temp1) temp1->prev = ptr2;

        if (prev2) prev2->next = ptr1;

    }

    else

    {

        ptr1->next = temp2;

        ptr1->prev = prev2;

        ptr2->next = temp1;

        ptr2->prev = prev1;

        if (temp1) temp1->prev = ptr2;

        if (prev1) prev1->next = ptr2;

        if (temp2) temp2->prev = ptr1;

        if (prev2) prev2->next = ptr1;

    }

    head->prev = tail;

    tail->next = head;

}

void solve(Node \*&head, Node \*&tail, int pos1, int pos2)

{

    if (pos1 == pos2) return;

    Node \*ptr1 = head;

    Node \*ptr2 = head;

    int length = 1;

    while (ptr1->next != head)

    {

        length++;

        ptr1 = ptr1->next;

    }

    if (pos1 < 1 || pos2 < 1 || pos1 > length || pos2 > length)

    {

        cout << "Invalid" << endl;

        return;

    }

    ptr1 = head;

    ptr2 = head;

    for (int i = 1; i < pos1; i++)

    {

        ptr1 = ptr1->next;

    }

    for (int i = 1; i < pos2; i++)

    {

        ptr2 = ptr2->next;

    }

    swap(ptr1, ptr2, head, tail);

}

void print(Node \*head)

{

    if (!head) return;

    Node \*ptr = head;

    do

    {

        cout << ptr->val << " ";

        ptr = ptr->next;

    } while (ptr != head);

    cout << endl;

}

int main()

{

    Node \*head = nullptr;

    Node \*tail = nullptr;

    InsertTail(head, tail, 1);

    InsertTail(head, tail, 2);

    InsertTail(head, tail, 3);

    InsertTail(head, tail, 6);

    InsertTail(head, tail, 10);

    char choice;

    do

    {

        cout << "List before swap: ";

        print(head);

        int pos1, pos2;

        cout << "Enter position 1 and position 2: ";

        cin >> pos1 >> pos2;

        solve(head, tail, pos1, pos2);

        cout << "After swapping: ";

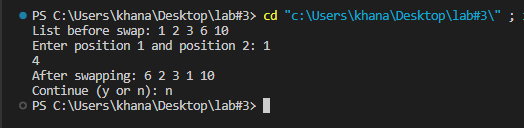
        print(head);

        cout << "Continue (y or n): ";

        cin >> choice;

    } while (choice == 'y');

}

****

**Quest # 4**

#include <iostream>

using namespace std;

class Node

{

public:

    int val;

    Node \*next;

    Node \*prev;

    Node(int data)

    {

        val = data;

        next = nullptr;

        prev = nullptr;

    }

};

void InsertTail(Node \*&head, Node \*&tail, int val)

{

    Node \*temp = new Node(val);

    if (!head)

    {

        head = temp;

        tail = temp;

        head->next = head;

        head->prev = head;

    }

    else

    {

        temp->prev = tail;

        temp->next = head;

        tail->next = temp;

        head->prev = temp;

        tail = temp;

    }

}

void reverse(Node \*&head, Node \*&tail)

{

    if (!head || !head->next)

    {

        return;

    }

    Node \*curr = head;

    Node \*temp = nullptr;

    do

    {

        temp = curr->prev;

        curr->prev = curr->next;

        curr->next = temp;

        curr = curr->prev;

    } while (curr != head);

    if (temp != nullptr)

    {

        tail = head;

        head = temp->prev;

    }

    head->prev = tail;

    tail->next = head;

}

void print(Node \*head, Node \*tail)

{

    Node \*temp = head;

    do

    {

        cout << temp->val << " ";

        temp = temp->next;

    } while (temp != head);

    cout << endl;

}

void solve(Node \*&head, Node \*&tail)

{

    Node \*temp\_head = nullptr;

    Node \*temp\_tail = nullptr;

    Node \*ptr = head;

    Node \*prev = nullptr;

    int count = 1;

    do

    {

        if (count % 2 == 0)

        {

            InsertTail(temp\_head, temp\_tail, ptr->val);

            Node \*to\_delete = ptr;

            prev->next = ptr->next;

            ptr->next->prev = prev;

            if (to\_delete == head)

                head = head->next;

            if (to\_delete == tail)

                tail = prev;

            ptr = ptr->next;

            delete to\_delete;

        }

        else

        {

            prev = ptr;

            ptr = ptr->next;

        }

        count++;

    } while (ptr != head);

    if (temp\_head)

    {

        reverse(temp\_head, temp\_tail);

        prev->next = temp\_head;

        temp\_head->prev = prev;

        temp\_tail->next = head;

        head->prev = temp\_tail;

    }

}

int main()

{

    Node \*head = nullptr;

    Node \*tail = nullptr;

    InsertTail(head, tail, 10);

    InsertTail(head, tail, 4);

    InsertTail(head, tail, 9);

    InsertTail(head, tail, 1);

    InsertTail(head, tail, 3);

    InsertTail(head, tail, 5);

    InsertTail(head, tail, 9);

    InsertTail(head, tail, 4);

    cout << "Original List: ";

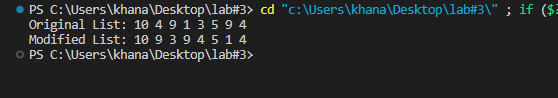
    print(head, tail);

    solve(head, tail);

    cout << "Modified List: ";

    print(head, tail);

}

****

**END**