1.Insert, Delete, and Display the element at position m in a double linked list.

insert

// Structure of the node

struct node

{

int data;

struct node \*next; // Pointer to next node

struct node \*prev; // Pointer to previous node

};

// Function to insert node with value as value1.

// The new node is inserted after the node with

// with value2

void insertAfter(struct Node\*\* start, int value1,

                                      int value2)

{

    struct Node\* new\_node = new Node;

    new\_node->data = value1; // Inserting the data

    // Find node having value2 and next node of it

    struct Node \*temp = \*start;

    while (temp->data != value2)

        temp = temp->next;

    struct Node \*next = temp->next;

    // insert new\_node between temp and next.

    temp->next = new\_node;

    new\_node->prev = temp;

    new\_node->next = next;

    next->prev = new\_node;

}

<https://www.geeksforgeeks.org/doubly-circular-linked-list-set-1-introduction-and-insertion/>

<https://www.softwaretestinghelp.com/doubly-linked-list/>

delete

/\* a node of the doubly linked list \*/

struct Node {

    int data;

    struct Node\* next;

    struct Node\* prev;

};

/\* Function to delete a node in a Doubly Linked List.

   head\_ref --> pointer to head node pointer.

   del  -->  pointer to node to be deleted. \*/

void deleteNode(struct Node\*\* head\_ref, struct Node\* del)

{

    /\* base case \*/

    if (\*head\_ref == NULL || del == NULL)

        return;

    /\* If node to be deleted is head node \*/

    if (\*head\_ref == del)

        \*head\_ref = del->next;

    /\* Change next only if node to be deleted is NOT the last node \*/

    if (del->next != NULL)

        del->next->prev = del->prev;

    /\* Change prev only if node to be deleted is NOT the first node \*/

    if (del->prev != NULL)

        del->prev->next = del->next;

    /\* Finally, free the memory occupied by del\*/

    free(del);

    return;

}

Display

|  |
| --- |
| //Prints all the elements in linked list in forward traversal order |
|  | void Print() { |
|  | struct Node\* temp = head; |
|  | printf("Forward: "); |
|  | while(temp != NULL) { |
|  | printf("%d ",temp->data); |
|  | temp = temp->next; |
|  | } |
|  | printf("\n"); |
|  | } |
|  |  |
|  | //Prints all elements in linked list in reverse traversal order. |
|  | void ReversePrint() { |
|  | struct Node\* temp = head; |
|  | if(temp == NULL) return; // empty list, exit |
|  | // Going to last Node |
|  | while(temp->next != NULL) { |
|  | temp = temp->next; |
|  | } |
|  | // Traversing backward using prev pointer |
|  | printf("Reverse: "); |
|  | while(temp != NULL) { |
|  | printf("%d ",temp->data); |
|  | temp = temp->prev; |
|  | } |
|  | printf("\n"); |
|  | } |

2. Swap the first and last nodes in a double linked list (Do Not Swap Data).

3. Reverse a double linked list in place.

/\* Program to reverse a doubly linked list \*/

#include <stdio.h>

#include <stdlib.h>

/\* a node of the doubly linked list \*/

struct Node

{

  int data;

  struct Node \*next;

  struct Node \*prev;

};

/\* Function to reverse a Doubly Linked List \*/

void reverse(struct Node \*\*head\_ref)

{

     struct Node \*temp = NULL;

     struct Node \*current = \*head\_ref;

     /\* swap next and prev for all nodes of

       doubly linked list \*/

     while (current !=  NULL)

     {

       temp = current->prev;

       current->prev = current->next;

       current->next = temp;

       current = current->prev;

     }

     /\* Before changing head, check for the cases like empty

        list and list with only one node \*/

     if(temp != NULL )

        \*head\_ref = temp->prev;

}

<https://www.geeksforgeeks.org/reverse-a-doubly-linked-list/>

4. Check if a circular double linked list is sorted or not.

/\* Linked list node \*/

struct Node

{

    int data;

    struct Node\* next;

};

// function to Check Linked List is

// sorted in descending order or not

bool isSortedDesc(struct Node \*head)

{

    if (head == NULL)

        return true;

    // Traverse the list till last node and return

    // false if a node is smaller than or equal

    // its next.

    for (Node \*t=head; t->next != NULL; t=t->next)

       if (t->data <= t->next->data)

            return false;

    return true;

}

Node \*newNode(int data)

{

   Node \*temp = new Node;

   temp->next = NULL;

   temp->data = data;

}

// Driver program to test above

int main()

{

    struct Node \*head = newNode(7);

    head->next = newNode(5);

    head->next->next = newNode(4);

    head->next->next->next = newNode(3);

    isSortedDesc(head) ? cout << "Yes" :

                         cout << "No";

    return 0;

}

<https://www.geeksforgeeks.org/check-linked-list-sorting-order/>

5. Display all elements above the average in a circular double linked list.

// Structure for a node

struct Node {

    int data;

    struct Node\* next;

};

// Function to insert a node at the beginning

// of a Circular linked list

void push(struct Node\*\* head\_ref, int data)

{

    struct Node\* ptr1 = (struct Node\*)malloc(sizeof(struct Node));

    struct Node\* temp = \*head\_ref;

    ptr1->data = data;

    ptr1->next = \*head\_ref;

    // If linked list is not NULL then

    // set the next of last node

    if (\*head\_ref != NULL) {

        while (temp->next != \*head\_ref)

            temp = temp->next;

        temp->next = ptr1;

    }

    else

        ptr1->next = ptr1; // For the first node

    \*head\_ref = ptr1;

}

// Function to find sum of the given

// Circular linked list

int sumOfList(struct Node\* head)

{

    struct Node\* temp = head;

    int sum = 0;

    if (head != NULL) {

        do {

            temp = temp->next;

            sum += temp->data;

        } while (temp != head);

    }

    return sum;

}

6. Insert an element in a sorted circular double linked list.

// Doubly linked list node

struct node {

    int data;

    struct node\* next;

    struct node\* prev;

};

// Utility function to create a node in memory

struct node\* getNode()

{

    return ((struct node\*)malloc(sizeof(struct node)));

}

// Function to display the list

int displayList(struct node\* temp)

{

    struct node\* t = temp;

    if (temp == NULL)

        return 0;

    else {

        cout << "The list is: ";

        while (temp->next != t) {

            cout << temp->data << " ";

            temp = temp->next;

        }

        cout << temp->data << endl;

        return 1;

    }

}

// Function to count nunmber of

// elements in the list

int countList(struct node\* start)

{

    // Declare temp pointer to

    // traverse the list

    struct node\* temp = start;

    // Variable to store the count

    int count = 0;

    // Iterate the list and increment the count

    while (temp->next != start) {

        temp = temp->next;

        count++;

    }

    // As the list is circular, increment the

    // counter at last

    count++;

    return count;

}

// Function to insert a node at a given position

// in the circular doubly linked list

bool insertAtLocation(struct node\* start, int data, int loc)

{

    // Declare two pointers

    struct node \*temp, \*newNode;

    int i, count;

    // Create a new node in memory

    newNode = getNode();

    // Point temp to start

    temp = start;

    // count of total elements in the list

    count = countList(start);

    // If list is empty or the position is

    // not valid, return false

    if (temp == NULL || count < loc)

        return false;

    else {

        // Assign the data

        newNode->data = data;

        // Iterate till the loc

        for (i = 1; i < loc - 1; i++) {

            temp = temp->next;

        }

        // See in Image, circle 1

        newNode->next = temp->next;

        // See in Image, Circle 2

        (temp->next)->prev = newNode;

        // See in Image, Circle 3

        temp->next = newNode;

        // See in Image, Circle 4

        newNode->prev = temp;

        return true;

    }

    return false;

}

// Function to create circular doubly linked list

// from array elements

void createList(int arr[], int n, struct node\*\* start)

{

    // Declare newNode and temporary pointer

    struct node \*newNode, \*temp;

    int i;

    // Iterate the loop until array length

    for (i = 0; i < n; i++) {

        // Create new node

        newNode = getNode();

        // Assign the array data

        newNode->data = arr[i];

        // If it is first element

        // Put that node prev and next as start

        // as it is circular

        if (i == 0) {

            \*start = newNode;

            newNode->prev = \*start;

            newNode->next = \*start;

        }

        else {

            // Find the last node

            temp = (\*start)->prev;

            // Add the last node to make them

            // in circular fashion

            temp->next = newNode;

            newNode->next = \*start;

            newNode->prev = temp;

            temp = \*start;

            temp->prev = newNode;

        }

    }

}

// Driver Code

int main()

{

    // Array elements to create

    // circular doubly linked list

    int arr[] = { 1, 2, 3, 4, 5, 6 };

    int n = sizeof(arr) / sizeof(arr[0]);

    // Start Pointer

    struct node\* start = NULL;

    // Create the List

    createList(arr, n, &start);

    // Display the list before insertion

    displayList(start);

    // Inserting 8 at 3rd position

    insertAtLocation(start, 8, 3);

    // Display the list after insertion

    displayList(start);

    return 0;

}

<https://www.geeksforgeeks.org/insertion-at-specific-position-in-a-circular-doubly-linked-list/>

7. Display all prime elements in a circular double linked list.

void display(struct Node\* start)

{

    struct Node \*temp = start;

    printf("\nTraversal in forward direction \n");

    while (temp->next != start)

    {

        printf("%d ", temp->data);

        temp = temp->next;

    }

    printf("%d ", temp->data);

    printf("\nTraversal in reverse direction \n");

    Node \*last = start->prev;

    temp = last;

    while (temp->prev != last)

    {

        printf("%d ", temp->data);

        temp = temp->prev;

    }

    printf("%d ", temp->data);

}

8. Check if a circular double linked list is in mirror format or not.

    insertAtLocation(start, 8, 3);

9. Dispose a double linked list.

/\* Link list node \*/

struct Node

{

    int data;

    struct Node\* next;

};

/\* Function to delete the entire linked list \*/

void deleteList(struct Node\*\* head\_ref)

{

   /\* deref head\_ref to get the real head \*/

   struct Node\* current = \*head\_ref;

   struct Node\* next;

   while (current != NULL)

   {

       next = current->next;

       free(current);

       current = next;

   }

   /\* deref head\_ref to affect the real head back

      in the caller. \*/

   \*head\_ref = NULL;

}

/\* Given a reference (pointer to pointer) to the head

  of a list and an int, push a new node on the front

  of the list. \*/

void push(struct Node\*\* head\_ref, int new\_data)

{

    /\* allocate node \*/

    struct Node\* new\_node =

            (struct Node\*) malloc(sizeof(struct Node));

    /\* put in the data  \*/

    new\_node->data  = new\_data;

    /\* link the old list off the new node \*/

    new\_node->next = (\*head\_ref);

    /\* move the head to point to the new node \*/

    (\*head\_ref)    = new\_node;

}

/\* Driver program to test count function\*/

int main()

{

    /\* Start with the empty list \*/

    struct Node\* head = NULL;

    /\* Use push() to construct below list

     1->12->1->4->1  \*/

    push(&head, 1);

    push(&head, 4);

    push(&head, 1);

    push(&head, 12);

    push(&head, 1);

    printf("\n Deleting linked list");

    deleteList(&head);

    printf("\n Linked list deleted");

}