Day 119/180 Doubly Linked List

1: Given an array, convert it into a doubly linked List by inserting each element at the start of the Doubly Linked List. Solve the problem iteratively and Recursively.

The iterative method is explained in the video.

Recursive method :

```cpp
#include <iostream>

using namespace std;
// Define Node class for doubly linked list
class Node {
public:
    int data;
    Node* next;
    Node* prev;

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

// Recursive function to convert array to doubly linked list
Node* arrayToDoublyLinkedListRecursive(int arr[], int size, int index
= 0) {
    if (index == size) {
        return nullptr;  // Base case: end of the array
    }

    Node* newNode = new Node(arr[index]);
    Node* restOfList = arrayToDoublyLinkedListRecursive(arr, size,
index + 1);

    newNode->next = restOfList;
    if (restOfList != nullptr) {
        restOfList->prev = newNode;
    }

    return newNode;
}

// Function to print doubly linked list
void printDoublyLinkedList(Node* head) {
    while (head != nullptr) {
        cout << head->data << " ";
        head = head->next;
    }
    cout << endl;
}

int main() {
    int arr[] = {1, 2, 3, 4, 5};
    int size = sizeof(arr) / sizeof(arr[0]);

    // Recursive approach
    Node* headRecursive = arrayToDoublyLinkedListRecursive(arr,
size);
    cout << "Doubly Linked List (Recursive): ";
    printDoublyLinkedList(headRecursive);

    return 0;
}
```

The code creates a doubly linked list from an array using a recursive approach. It defines a Node class for the elements of the list, and a recursive function arrayToDoublyLinkedListRecursive to build the list. The printDoublyLinkedList function prints the resulting list. The main function demonstrates the conversion of an array to a doubly linked list and prints the list using the recursive approach.

2:Doubly linked list Insertion at given position:

The addNode function inserts a new node with a specified data at a given position in a doubly linked list. It traverses the list to the specified position, then inserts the new node by updating pointers accordingly.

```
void addNode(Node *head, int pos, int data)
{
   // Your code here
   Node *newNode = new Node(data);

   while(pos > 0)
   {
       head = head->next;
       pos--;
   }

   Node *temp = head->next;
   head->next = newNode;
   newNode->prev = head;
   newNode->next = temp;
   if(temp != NULL)
   temp->prev = newNode;

}
```

3: Reverse a Doubly Linked List:

The reverseDLL function takes a doubly linked list and reverses its direction. It uses two pointers (curr and newHead) to traverse and reverse the list in-place.

The while loop iterates through the list, swapping next and prev pointers for each node. Finally, it returns the new head of the reversed list.

```
Node* reverseDLL(Node * head)
  {
      //Your code here
      if(head==NULL)
  {
      return NULL;
  }


 Node* curr=head;
 Node* newHead=NULL;

 while(curr!=NULL)
 {
    Node* currNext=curr->next;
    curr->next=curr->prev;
    curr->prev=currNext;
    newHead=curr;
    curr=currNext;
 }
 return newHead;
  }
```