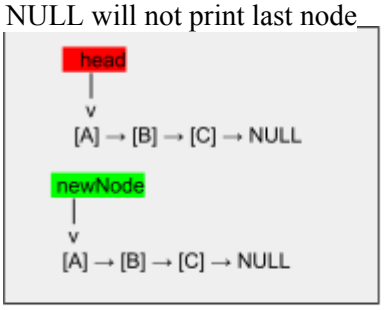
Labsheet 2

Question 01

```
#include <stdio.h>
#include <stdlib.h>

// Define the structure for a linked list node
struct Node {
    int data;
    struct Node *Next;
};

struct Node *head = NULL;// Global head pointer

// Function to create a new node
struct Node* createNode(int data) {
    struct Node *newNode = (struct Node*)malloc(sizeof(struct Node));

    newNode->data = data;
    newNode->Next = NULL;

    return newNode;
}

// Function to print the linked list
void PrintSinglyLinkedList() {
    struct Node *temp = head;

    if (head == NULL) {
        printf("List is Empty\n");
        return;
    }

    printf("Linked List: ");
    while (temp != NULL) { //here we cant use  temp -> Next != NULL will not print last node_
        printf("%d -> ", temp->data);
        temp = temp->Next;
    }
    printf("NULL\n");
}

// Function to insert a node at the beginning
void InsertAtBeginning(int data) {
    struct Node *newNode = createNode(data);
    newNode->Next = head;
    head = newNode;
}

// Function to insert a node at a given position
void InsertAtGivenPosition(int position, int data) {
    struct Node *newNode = createNode(data);
    if (position == 1) {
        InsertAtBeginning(data);
        return;
    }
    struct Node *temp = head;
    for (int i = 1; temp != NULL && i < position - 1; i++) {
        temp = temp->Next;
    }
    if (temp == NULL) {
```

`temp != NULL`:
Checks that the **pointer itself is valid**. It processes every node, including the last one.

`temp->next != NULL`:
Checks that there is another node after the current one. **This stops the loop before the last node, so the last node is not printed.**



Position: 1 2 3 4
Data: 10 20 30 40

The linked list looks like:

head --> [10] --> [20] --> [30] --> [40] --> NULL

Pos = 3

Iteration 1
Int i =1; temp != NULL && 1 < 2
temp = temp->Next (node 1 -> node 2)

Iteration 2
Now i = 2 ; **Check: 2 < 2 → false (Loop stops)**
No any action

newNode->Next = temp->Next;
temp->Next = newNode;

(I insert the new node (say, containing data X) after the node with data 20 and before the node with data 30.)

head --> [10] --> [20] --> [X] --> [30] --> [40] --> NULL

```
        printf("Invalid position!\n");
        return;
    }

    newNode->Next = temp->Next;
    temp->Next = newNode;
}

// Function to insert a node at the end
void InsertAtEnd(int data) {
    struct Node *newNode = createNode(data);

    if (head == NULL) {
        head = newNode;
        return;
    }

    struct Node *temp = head;
    while (temp->Next != NULL) {
        temp = temp->Next;
    }
    temp->Next = newNode;
}

// Function to delete a node from the beginning
void DeleteAtBeginning() {
    if (head == NULL) {
        printf("List is Empty!\n");
        return;
    }

    struct Node *temp = head;
    head = head->Next;
    free(temp);
}

// Function to delete a node at a given position
void DeleteAtGivenPosition(int position) {
    if (head == NULL) {
        printf("List is Empty!\n");
        return;
    }

    struct Node *temp = head;

    if (position == 1) {
        head = temp->Next;
        free(temp);
        return;
    }

    struct Node *prev = NULL;
    for (int i = 1; temp != NULL && i < position; i++) {
        prev = temp;
        temp = temp->Next;
    }

    if (temp == NULL) {
        printf("Invalid position!\n");
        return;
    }
```

```
    prev->Next = temp->Next;
    free(temp);
}

// Function to delete a node from the end
void DeleteAtEnd() {
    if (head == NULL) {
        printf("List is Empty!\n");
        return;
    }

    struct Node *temp = head;
    struct Node *prev = NULL;

    if (head->Next == NULL) {
        free(head);
        head = NULL;
        return;
    }

    while (temp->Next != NULL) {
        prev = temp;
        temp = temp->Next;
    }

    prev->Next = NULL;
    free(temp);
}

// Main function to test the linked list operations
int main() {
    InsertAtBeginning(3);
    InsertAtBeginning(2);
    InsertAtBeginning(1);
    PrintSinglyLinkedList();

    InsertAtEnd(4);
    InsertAtEnd(5);
    PrintSinglyLinkedList();

    InsertAtGivenPosition(3, 10);
    PrintSinglyLinkedList();

    DeleteAtBeginning();
    PrintSinglyLinkedList();

    DeleteAtGivenPosition(3);
    PrintSinglyLinkedList();

    DeleteAtEnd();
    PrintSinglyLinkedList();

    return 0;
}
```

Question 02

```
//Insert At Last
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

struct Node
{
   int data;
   struct Node *Next;
};

struct Node *head = NULL;//First Node

bool IsEmpty()
{
   if(head == NULL)
   {
      return true;
   }
   return false;
}
void InserAtLastInSingleLinkedList(int data)
{
   struct Node *NewNode = (struct Node*)malloc(sizeof(struct Node));
   struct Node *Temp = head;

   NewNode ->data = data;
   NewNode ->Next = NULL;

   if(IsEmpty())
   {
      head = NewNode;
      return;
   }
   else
   {
      while(Temp->Next != NULL)
      {
         Temp = Temp->Next;
      }
      Temp->Next = NewNode;
   }
}
```

```
void PrintLinkedList()
{
   struct Node *temp = head;

   if(IsEmpty())
   {
      printf("List is Empty");
      return;
   }
   else
   {
      while(temp!= NULL)
      {
         printf("%d -> ", temp->data);
         temp = temp->Next;
      }
      printf("%s\n" ,"NULL");
      return;
   }
}

int main()
{
   InserAtLastInSingleLinkedList(2);
   InserAtLastInSingleLinkedList(3);
   InserAtLastInSingleLinkedList(4);

   printf("this is singly linked List\n\n");
   PrintLinkedList();
   return 0;
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

struct Node
{
   int data;
   struct Node *Next;
};

struct Node *head = NULL;//First Node

bool IsEmpty()
{
   return head == NULL;
}
void InserAtHeadInSingleLinkedList(int data)
{
   struct Node *NewNode = (struct Node*)malloc(sizeof(struct Node));

   NewNode ->data = data;
   NewNode ->Next = NULL;

   if(IsEmpty())
   {
      head = NewNode;
      return;
   }
   else
   {
      NewNode->Next = head;
      head = NewNode;
      return;
   }
}
void PrintLinkedList()
{
   struct Node *temp = head;

   if(IsEmpty())
   {
      printf("List is Empty");
      return;
   }
   else
   {
      while(temp!= NULL)
      {
         printf("%d -> ", temp->data);
         temp = temp->Next;
      }
      printf("%s\n","NULL");
      return;
   }
}
int main()
{
```

```
   InserAtHeadInSingleLinkedList(2);    InserAtHeadInSingleLinkedList(3);    InserAtHeadInSingleLinkedList(4);

   printf("this is singly linked List\n\n");
   PrintLinkedList();    return 0;}
//insert at given point
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

struct Node
{
   int data;
   struct Node *Next;
};

struct Node *head = NULL;//First Node

int Position =0;

bool IsEmpty()
{
   return head == NULL;
}
void InserAtLastInSingleLinkedList(int data)
{
   struct Node *NewNode = (struct Node*)malloc(sizeof(struct Node));
   struct Node *Temp = head;

   NewNode ->data = data;
   NewNode ->Next = NULL;

   if(IsEmpty())
   {
      head = NewNode;
      return;
   }
   else
   {
      while(Temp->Next != NULL)
      {
         Temp = Temp->Next;
      }
      Temp->Next = NewNode;
   }
}
void InserAtGivenPositionInSingleLinkedList(int Position, int data)
{
   struct Node *NewNode = (struct Node*)malloc(sizeof(struct Node));

   struct Node *temp = head;

   int count = 1;

   NewNode->data = data;
   NewNode->Next = NULL;

   // Case 1: Insert at Head if Position == 1
   if (Position == 1)
   {
      NewNode->Next = head;
      head = NewNode;
      return;
```

```
    }

    // Traverse to the node just before the desired position
    while (temp != NULL && count < Position - 1)
    {
        temp = temp->Next;
        count++;
    }

    // If Position is out of bounds
    if (temp == NULL)
    {
        printf("Position out of range\n");
        return;
    }

    // Insert NewNode in the middle
    NewNode->Next = temp->Next;
    temp->Next = NewNode;
}


void PrintLinkedList()
{
    struct Node *temp = head;

    if(IsEmpty())
    {
        printf("List is Empty");
        return;
    }
    else
    {
        printf("\t");
        while(temp!= NULL)
        {

            printf("%d -> ", temp->data);
            temp = temp->Next;
        }
        printf("%s\n","NULL");
        return;
    }
}


int main()
{
    InserAtLastInSingleLinkedList(2);
    InserAtLastInSingleLinkedList(3);
    InserAtLastInSingleLinkedList(4);


    printf("This is singly linked List\n\n");
    PrintLinkedList();

    printf("\nThis is after inserting given position singly linked List\n\n");
    InserAtGivenPositionInSingleLinkedList(2,22);
    PrintLinkedList();


    return 0;
```

```
}
```

Part II

Deletion

```
#include <stdio.h>  // Standard input-output library
#include <stdlib.h> // Standard library for memory allocation
#include <stdbool.h> // Boolean data type support

// Define a structure for a node in the linked list
struct Node {
    int data; // Data field to store integer value
    struct Node *Next; // Pointer to the next node in the list
};

struct Node *head = NULL; // Head pointer to the first node, initially NULL

// Function to check if the linked list is empty
bool IsEmpty() {
    return head == NULL; // Returns true if head is NULL (list is empty)
}

// Function to insert a node at the end of the linked list
void InserAtLastInSingleLinkedList(int data) {
    struct Node *NewNode = (struct Node*)malloc(sizeof(struct Node)); // Allocate memory for new node
    struct Node *Temp = head; // Temporary pointer to traverse the list
    NewNode->data = data; // Assign data to the new node
    NewNode->Next = NULL; // Set next pointer to NULL as it's the last node

    if (IsEmpty()) { // If list is empty, set head to new node
        head = NewNode;
        return;
    }

    while (Temp->Next != NULL) { // Traverse to the last node
        Temp = Temp->Next;
    }
    Temp->Next = NewNode; // Link last node to new node
}

// Function to delete the first node in the linked list
void DeleteAtBeginning() {
    if (IsEmpty()) { // If list is empty, print message
        printf("List is already empty.\n");
        return;
    }
    struct Node *Temp = head; // Temporary pointer to hold the first node
    head = head->Next; // Move head to the next node
    free(Temp); // Free memory of deleted node
}

// Function to delete a node at a given position
void DeleteAtPosition(int position) {
    if (IsEmpty() || position < 1) { // Check for invalid position or empty list
        printf("Invalid position or empty list.\n");
        return;
    }

    struct Node *Temp = head; // Temporary pointer to traverse the list
    if (position == 1) { // If deleting the first node
        head = head->Next;
```

```
        free(Temp); // Free memory of deleted node
        return;
    }

    struct Node *Prev = NULL; // Pointer to keep track of previous node
    for (int i = 1; Temp != NULL && i < position; i++) { // Traverse to the position
        Prev = Temp;
        Temp = Temp->Next;
    }

    if (Temp == NULL) { // If position exceeds list size
        printf("Position exceeds list size.\n");
        return;
    }

    Prev->Next = Temp->Next; // Unlink the node from the list
    free(Temp); // Free memory of deleted node
}

// Function to delete the last node in the linked list
void DeleteAtEnd() {
    if (IsEmpty()) { // If list is empty, print message
        printf("List is already empty.\n");
        return;
    }

    struct Node *Temp = head; // Temporary pointer to traverse the list
    struct Node *Prev = NULL; // Pointer to keep track of previous node

    if (head->Next == NULL) { // If only one node exists
        free(head); // Free memory of the single node
        head = NULL; // Set head to NULL
        return;
    }

    while (Temp->Next != NULL) { // Traverse to the last node
        Prev = Temp;
        Temp = Temp->Next;
    }

    Prev->Next = NULL; // Unlink last node
    free(Temp); // Free memory of deleted node
}

// Function to print the linked list
void PrintLinkedList() {
    struct Node *temp = head; // Temporary pointer to traverse the list
    if (IsEmpty()) { // If list is empty, print message
        printf("List is Empty\n");
        return;
    }
    while (temp != NULL) { // Traverse through the list
        printf("%d -> ", temp->data); // Print node data
        temp = temp->Next; // Move to next node
    }
    printf("NULL\n"); // Indicate end of list
}

// Main function to demonstrate operations
int main() {
    InserAtLastInSingleLinkedList(2); // Insert nodes at the end
    InserAtLastInSingleLinkedList(3);
```

```
    InserAtLastInSingleLinkedList(4);
    InserAtLastInSingleLinkedList(5);

    printf("Original singly linked list:\n");
    PrintLinkedList(); // Print initial list

    DeleteAtBeginning(); // Delete first node
    printf("After deleting first node:\n");
    PrintLinkedList();

    DeleteAtPosition(2); // Delete node at position 2
    printf("After deleting node at position 2:\n");
    PrintLinkedList();

    DeleteAtEnd(); // Delete last node
    printf("After deleting last node:\n");
    PrintLinkedList();

    return 0; // End of program
}
```