Assignment 4

1 Write a function "insert_any()" for inserting a node at a given position of linked list.
Assume position starts at 0.

Inserting a node

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

struct Node
{
    int data;
    struct Node *next;
};
struct Node *head;
void insert (int data, int n)
{
    Node *temp1 = new node ();
    temp1->data = data;
    temp1 -> next = NULL;
    if (n==1) {
        temp -> next = head;
        head = temp1;
        return;
    }
    Node* temp2 = new Node ();
    for (int i=0; i<n-2; i++)
    {
        temp2 = temp2-> next;
    }
    temp1-> next = temp2-> next;
    temp2 ->next = temp2;
}
```

```
void print()
{
    Node * temp = head;
    while (temp != NULL){
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}
int main()
{
    head = NULL;
    Insert(7,1)
    Insert(5,2)
    Insert(8,1)
    Insert(6,2)
    print();
System("pause");
```

Question 2 - Write a function "delete_beg" for deleting a node from beginning of the linked list.

PROGRAM

```
#include <iostream.h>
using namespace std;

Struct Node {
        int data;
        Struct Node * next;
};
Node * remove FirstNode (struct Node * head)
{       if (head == NULL)
            return NULL;
    // move the head pointer to the next node
        Node * temp = head;
        head = head -> next;
        delete temp;
        return head;
}
void push (struct Node ** head_ref, int new_data)
{
            Struct Node * node = newNode;
            new node -> data = new_data;
             new_ node -> next = (head * _ref);
```

```
    (*head_ref) = new_node;
}

int main()
{
    Node* head = NULL;

        push (&head, 12);
        push (&head, 29);
        push (&head, 11);
        push (&head, 23);
        push (&head, 8);

        head = removeFirstNode(head);
        for (Node* temp = head; temp != NULL
        cout << temp->data << " ";

        return 0;

}
```

Output

23 11 29 12

3→ Write a function "delete_end()" for deleting a node from the end of the linked list.

```
using namespace std;

class Node {
  Public :
      int data;
      Node* next;
};

void push (Node** head_ref , int new_data)
{
      Node * new_node = new Node ();
      new Node ->data = new_data;
      new - node → next = (* head_ref);
      *head _ref) = new node
      }
void delete node(Node** head_ref, int key)
 {
      Node * temp = *head_ref;
      Node * prev = NULL;
     if (temp != NULL && temp -> data == key)
     {   *head_ref = temp -> next;
         delete temp;
         return;
     }
```

```
while (temp != NULL && temp->data != key)
{
    prev = temp;
    temp = temp->next;
}
if (temp == NULL)                    // If key was not present in Linked list
    return;
// Unlink the node from linked list
prev->next = temp->next;

// free memory
delete temp;
}
void printList(Node* node)
{
    while (node != NULL)
    {
        cout << node->data << " ";
        node = node->next;
    }
}
// Driver code
int main()
{
    // Start with the empty list
    Node* head = NULL;
    // Add elements in linked list
    push(&head, 7);
    push(&head, 1);
    push(&head, 3);
    push(&head, 2);
```

```
puts ("Created linked list : ")
print list (head);

delete Node (&head, 1);
puts ("\n linked list after deletion of 1: ");

print list (head);
return 0;
}
```

Output

Created Linked list:

2 3 17

linked list after deletion of 1:

2 3 7

4 → In the binary search algorithm, it is suggested to calculate the mid as beg + (end-beg)/2 instead of (beg + end)/2. Why is it so?

Ans → In general case the both expressions are invalid. For example the first expression is invalid because there is no such operation as + for ~~for~~ pointers or iterators. The second expression is invalid in case when non-random access iterators are used

correct code is -

mid = std::next(beg, std::distance(beg, end)/2);

5 → Write the algorithm/function for ternary search.

Ans

~~Proc~~ function:

```
int ternary search (int l, int r, int x)
{
    if (r >= l)
    {
        int mid1 = l + (r-l)/3;
        int mid2 = r - (r-l)/3;
        if (ar[mid1] == x)
            return mid1;
        if (ar[mid2] == x)
            return mid2;
        if (x < ar[mid1])
            return
```

```
ternarySearch(l, mid1 - 1, x);
    else if (x > ar[mid2])
        return
ternarySearch(mid2 + 1, r, x);
    else
        return
    ternarySearch(mid1 + 1, mid2 - 1, x);

    }
    return -1;
}
```