Assignment 4 | 7th January 2021

Question 1

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

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

struct node *head=NULL;
struct node
        {
        int data;
        struct node *next;
        };
void ins(int data)
        {
        struct node *temp = (struct node*)malloc(sizeof(struct node));
        temp->data=data;
        temp->next=head;
        head=temp;
        }
void insert_any(int data,int position)
    {
        struct node *ptr = (struct node*)malloc(sizeof(struct node));
        ptr->data=data;
        int i;
        struct node *temp=head;
        if(position==1)
        {
            ptr->next=temp;
            head=ptr;
            return;
        }
        for(i=1;i<position-1;i++)
```

```
        {
            temp=temp->next;
        }

        ptr->next=temp->next;
        temp->next=ptr;
}
void display()
{
        struct node *temp=head;
        printf("\nList: ");
        while(temp!=NULL)
        {
            printf("\n%d ",temp->data);
            temp=temp->next;
    }
}
int main()
{
    int i, n, pos, data;
    printf("Enter the number of nodes: \n");
    scanf("%d",&n);
    printf("Enter the data for the nodes: \n");
    for(i=0;i<n;i++)
    {
        scanf("%d",&data);
        ins(data);
    }
    printf("Enter the data you want to insert in between the nodes: \n");
    scanf("%d",&data);
    printf("Enter the position at which you want to insert the nodes: \n");
    scanf("%d",&pos);
    if(pos>n)
    {
        printf("Enter a valid position: ");

    }
    else
    {
        insert_any(data,pos);

    }
    display();
    return 0;
}
```

Question 2

Write a function "delete_beg()" for deleting a node from the beginning of the linked list.

```c
#include <stdio.h>
#include <stdlib.h>
struct node {
    int data;
    struct node *next;
}*head;
void createList(int n);
void deleteFirstNode();
void displayList();

int main()
{
    int n, choice;

    printf("Enter the total number of nodes: ");
    scanf("%d", &n);
    createList(n);

    printf("\nData in the list \n");
    displayList();

    printf("\nPress 1 to delete first node: ");
    scanf("%d", &choice);

    if(choice == 1)
        delete_beg();

    printf("\nData in the list \n");
    displayList();

    return 0;
}


void createList(int n)
{
    struct node *newNode, *temp;
    int data, i;
```

```
    head = (struct node *)malloc(sizeof(struct node));

    if(head == NULL)
    {
        printf("Unable to allocate memory.");
    }
    else
    {

        printf("Enter the data of node 1: ");
        scanf("%d", &data);

        head->data = data;
        head->next = NULL;
        temp = head;

        for(i=2; i<=n; i++)
        {
            newNode = (struct node *)malloc(sizeof(struct node));

            if(newNode == NULL)
            {
                printf("Unable to allocate memory.");
                break;
            }
            else
            {
                printf("Enter the data of node %d: ", i);
                scanf("%d", &data);

                newNode->data = data;
                newNode->next = NULL;
                temp->next = newNode;
                temp = temp->next;
            }
        }

        printf("SINGLY LINKED LIST CREATED SUCCESSFULLY\n");
    }
}

void delete_beg()
{
    struct node *toDelete;
```

```
    if(head == NULL)
    {
        printf("List is already empty.");
    }
    else
    {
        toDelete = head;
        head = head->next;

        printf("\nData deleted = %d\n", toDelete->data);

        free(toDelete);

        printf("SUCCESSFULLY DELETED FIRST NODE FROM LIST\n");
    }
}

void displayList()
{
    struct node *temp;

    if(head == NULL)
    {
        printf("List is empty.");
    }
    else
    {
        temp = head;
        while(temp != NULL)
        {
            printf("Data = %d\n", temp->data);
            temp = temp->next;
        }
    }
}
```

Question 3

Write a function “delete_end()” for deleting a node from the end of the linked list.

Question 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?

```c
// program for calculating mid of array
#include <stdio.h>
#include <limits.h>
int main()
{
    int start = INT_MAX, end = INT_MAX;
    printf("start = %d \n", start);
    printf("end = %d \n", end);

    // method 1
    int mid1 = (start + end) / 2;
    printf("mid using (start + end)/2 = %d \n", mid1);

    // method 2
    int mid2 = start + (end - start) / 2;
    printf("mid using start + (end - start)/2 = %d \n", mid2);
    return 0;
}
```

- **Reason: -** find middle index of array once you know start index and end index of array, but there are certain benefits of using **start + (end – start)/2** over **(start + end)/2**, which are described below:
  The very first way of finding middle index is

  mid = (start + end)/2
  But there is problem with this approach, what if value of start or end or both is **INT_MAX**, it will cause integer overflow.
  The better way of calculating mid index is:

  mid = start + (end - start)/2

Question 5

Write the algorithm/function for Ternary Search.

```c
#include <stdio.h>
int ternarySearch(int array[], int left, int right, int x)
{
   if (right >= left) {
     int intvl = (right - left) / 3;
```

```
    int leftmid = left + intvl;
    int rightmid = leftmid + intvl;
    if (array[leftmid] == x)
       return leftmid;
    if (array[rightmid] == x)
       return rightmid;
    if (x < array[leftmid]) {
      return ternarySearch(array, left, leftmid, x);
    }
    else if (x > array[leftmid] && x < array[rightmid]) {
      return ternarySearch(array, leftmid, rightmid, x);
    }
    else {
      return ternarySearch(array, rightmid, right, x);
    }
  }
  return -1;
}
int main(void)
{
  int array[] = {1, 2, 3, 5};
  int size = sizeof(array)/ sizeof(array[0]);
  int find = 3;
  printf("Position of %d is %d\n", find, ternarySearch(array, 0, size-1, find));
  return 0;
}
```

- **Algorithm**

  The steps involved in this algorithm are:

- ✓ **Step 1**: Divide the search space (initially, the list) in three parts (with two mid-points: mid1 and mid2)
- ✓ **Step 2**: The target element is compared with the edge elements that is elements at location mid1, mid2 and the end of the search space. If element matches, go to step 3 else predict in which section the target element lies. The search space is reduced to 1/3rd. If the element is not in the list, go to step 4 or to step 1.
- ✓ **Step 3**: Element found. Return index and exit.
- ✓ **Step 4**: Element not found. Exit.

- **Complexity**

- Worst case time complexity: O(log N)
- Average case time complexity: O(log N)
- Best case time complexity: O(1)
- Space complexity: O(1)