Data Structures and Algorithms Essential Program

DAY-4 | ASSIGNMENT **Email i'd: pavanibhavya77@gmail.com**

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

```
#include<stdio.h>
#include<stdlib.h>
struct node{
   int data;
   struct noded* next;
}*start=NULL,*q,*t;
int main(){
   int ch;
   int insert_any();
   int print();
while(1){
   printf("\n1.insert at a position\n2.print\n");
   printf("enter choice");
   scanf("%d",&ch);
   switch(ch){
      case 1:insert_any();
      break;
      case 2:print();
      break;
      default:printf("wrong choice");
      break;
   }
}
}
   int insert_any(){
      int pos,i,num;

      t=(struct node*)malloc(sizeof (struct node ));

      printf("enter data");
      scanf("%d",&num);
      printf("enter pos");
      scanf("%d",&pos);
      if(start==NULL){
        t->next=NULL;
        start=t;
```

```
        }
        else{
        t->data=num;
        q=start;
        for(i=1;i<pos-1;i++){
            q=q->next;
        }
        t->next=q->next;
        q->next=t;
        return 0;
    }
    }
    int print(){
        if(start==NULL){
            printf("list is empty");
        }
        else{
            q=start;
            printf("the linked list is:\n");
            while(q!=NULL){
                printf("%d->",q->data);
                q=q->next;
            }
        }
    }
```

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

Sol.)

```
#include<stdio.h>
#include<stdlib.h>
struct node{
    int data;
    struct noded* next;
}*start=NULL,*q,*t;
int main(){
    int delete_beg();
}
void delete_beg(){
        if(start==NULL){
                printf(“list is empty”);
        }
        else{
                q=start;
```

```
                start=start->next;
                pritnf("deleted %d",q->data);
                free(q);
        }
}
```

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

```
#include<stdio.h>
#include<stdlib.h>
struct node{
   int data;
   struct noded* next;
}*start=NULL,*q,*t;
int main(){
   int delete_end();
}
void delete_end(){
        if(start==NULL){
                printf("list is empty");
        }
        else{
                q=start;
                while(q->next!=NULL){
                q=q->next;
                t=q->next;
                q->next=NULL;
        }
        printf("deleted %d",t->data);
        free(t);
}
```

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?

Sol.) there are 2 advantages of using beg+(end-beg)/2 to calculate the mid value in an array.

- ➔ If start and end values in an array are both INT_MAX then it gives a garbage value as output when **(beg + end) / 2** is used to calculate the mid.
  If we use **beg + (end - beg) / 2** to calculate the mid value when both start and end values are INT_MAX then we get accurate output.
  Therefore **beg + (end - beg) / 2** is preferred over **(beg + end) / 2.**
- ➔ **beg + (end - beg) / 2** can be used with pointers as pointer difference is valid in C and pointer addition is not possible.

5.)Write the algorithm/function for Ternary Search.

Sol.) In ternary search the array is divided into 3 parts

Algorithm:

- ➔ Start
- ➔ Input array and element to be searched
- ➔ Divide the array into 3 parts by taking 3 mid values
  Where
  mid1 = start + (end-start)/3
  mid2 = end – (end-start)/3
- ➔ If mid1==element
  then return the position of mid 1
- ➔ If mid 2==element
  then return the position of mid 2
- ➔ If element <mid1
  then search left part of the mid1 and return position of element if found
- ➔ If element >mid2
  Then search right part of mid 2 and return position of element if found
- ➔ Else search the right part of mid1/left part of mid2 and return position of element if found
- ➔ If element not found then exit
- ➔ Stop