1. Write a function “insert\_any()” for inserting a node at any given position of the linked list. Assume position starts at 0.
2. #include<stdio.h>

typedef int cat;

typedef struct node

{

int data;

struct node \*next;

}node;

node \*create(int x);

node \*insert\_any(node \*head, int x);

{

int main()

{

node \*head=NULL;

insert\_any(head,10);

}

node \*create(int x)

{

node \*new=(node \*)malloc(sizeof(node));

new -> data=x;

new -> next=NULL;

return new;

}

node \* insert\_any(node \*head,int x)

{

node \*new=create(x);

node \*temp=head;

if(!head)

{

head=new;

return head;

}

while(temp->next)

{

temp=temp->next;

}

temp->next=new;

return head;

}

node \*insert\_any(node \*n, int x,int pos)

{

node \*new=create(x)

new->node=temp;

new->data=x;

new->next=pos+1

if(temp->next==pos)

{

insert\_any(node \*n, int x,int pos)

}

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

1. node \*delete\_beg(node \*\*new -> next, int x, int pos)

{

node \*temp= \*new->next, \*prev;

if(temp!=NULL && temp->data!=x)

{

\*new->next=temp->next;

free(temp);

return;

}

while(temp!=NULL && temp->data!=x)

{

prev=temp;

temp=temp->next;

}

if(temp==NULL) return;

prev->next=temp->next;

free(temp);

}

3)Write a function “delete\_end()” for deleting a node from the end of the linked list.

1. node \*delete\_end(node \*head)

{

if(head==NULL) return NULL;

if(head->next==NULL)

{

free(head);

return NULL;

}

node \*second\_last=head;

while(second\_last->next->next!=NULL)

second\_last=second\_last->next;

free(second\_last->next);

second\_last->next=NULL;

return head;

}

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?

1. Exactly. There's no guarantee that “beg+end” is representable; but in the second case the intermediate values, as well as the expected result, are no larger than “end”, so there is no danger of overflow.

The second form can also be used for affine types like pointers and other random-access iterators, which can be subtracted to give a distance, but not added together.

So, The mid value can be “beg + (end-beg) / 2” rather than “(beg + end) / 2”.

5) Write the algorithm/function for Ternary Search.

1. int Ternary\_Search(int l, int r, int key, int a[])

{

if (r >= l) {

// Find the mid1 and mid2

int mid1 = l + (r - l) / 3;

int mid2 = r - (r - l) / 3;

// Check if key is present at any mid

if (a[mid1] == key) {

return mid1;

}

if (a[mid2] == key) {

return mid2;

}

// Since key is not present at mid,

// check in which region it is present

// then repeat the Search operation

// in that region

if (key < a[mid1]) {

// The key lies in between l and mid1

return ternarySearch(l, mid1 - 1, key, a);

}

else if (key > a[mid2]) {

// The key lies in between mid2 and r

return ternarySearch(mid2 + 1, r, key, a);

}

else {

// The key lies in between mid1 and mid2

return ternarySearch(mid1 + 1, mid2 - 1, key, a);

}

}

// Key not found

return -1;

}