**Question 1**

**Write a function “insert\_any()” for inserting a node at any given position of the linked list. Assume position starts at 0.**

**Ans:**

**void insert\_any(Node\*\* current, int pos, int data)**

**{**

**if (pos < 1|| pos > size + 1)**

**cout << "Invalid position!" << endl;**

**else {**

**while (pos--) {**

**if (pos == 0) {**

**Node\* temp = getNode(data);**

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

**\*current = temp;**

**}**

**else**

**current = &(\*current)->next;**

**}**

**size++;**

**}**

**}**

**Question 2**

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

**Ans:**

**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);**

**/\* Clears the memory occupied by first node\*/**

**free(toDelete);**

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

**}**

**}**

**Question 3**

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

**Ans:**

**Node\* delete\_end(struct Node\* head)**

**{**

**if (head == NULL)**

**return NULL;**

**if (head->next == NULL) {**

**delete head;**

**return NULL;**

**}**

**Node\* second\_last = head;**

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

**second\_last = second\_last->next;**

**delete (second\_last->next);**

**second\_last->next = NULL;**

**return head;**

**}**

**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?**

**Ans:**

**Point 1: Think of predicates**

**In general for all these 4 cases (and also the normal binary search for equality), imagine them as a predicate. So what this means is that some of the values are meeting the predicate and some some failing. So consider for example this array with a target of 5: [1, 2, 3, 4, 6, 7, 8]. Finding the first number greater than 5 is basically equivalent of finding the first one in this array: [0, 0, 0, 0, 1, 1, 1].**

**Point 2: Search boundaries inclusive**

**I like to have both ends always inclusive. But I can see some people like start to be inclusive and end exclusive (on len instead of len -1). I like to have all the elements inside of the array, so when referring to a[mid] I don't think whether that will give me an array out of bound. So my preference: Go inclusive!!!**

**Point 3: While loop condition <=**

**So we even want to process the subarray of size 1 in the while loop, and when the while loop finishes there should be no unprocessed element. I really like this logic. It's always solid as a rock. Initially all the elements are not inspected, basically they are unknown. Meaning that everything in the range of [st = 0, to end = len - 1] are not inspected. Then when the while loop finishes, the range of uninspected elements should be array of size 0!**

**Point 4: Loop invariants**

**Since we defined start = 0, end = len - 1, invariants will be like this: Anything left of start is smaller than target. Anything right of end is greater than or equal to the target.**

**Point 5: The answer**

**Once the loop finishes, basically based on the loop invariants anything to the left of start is smaller. So that means that start is the first element greater than or equal to the target. Equivalently, anything to the right of end is greater than or equal to the target. So that means the answer is also equal to end + 1.**

**Question 5**

**Write the algorithm/function for Ternary Search.**

**Ans:**

**#include <iostream>**

**using namespace std;**

**int ternarySearch(int l, int r, int key, int ar[])**

**{**

**while (r >= l) {**

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

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

**if (ar[mid1] == key) {**

**return mid1;**

**}**

**if (ar[mid2] == key) {**

**return mid2;**

**}**

**if (key < ar[mid1]) {**

**r = mid1 - 1;**

**}**

**else if (key > ar[mid2]) {**

**l = mid2 + 1;**

**}**

**else {**

**l = mid1 + 1;**

**r = mid2 - 1;**

**}**

**}**

**return -1;**

**}**

**int main()**

**{**

**int l, r, p, key;**

**int ar[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };**

**l = 0;**

**r = 9;**

**key = 5;**

**p = ternarySearch(l, r, key, ar);**

**cout << "Index of "<<key<<" is " << p << endl;**

**key = 50;**

**p = ternarySearch(l, r, key, ar);**

**cout << "Index of "<<key<<" is " << p;**

**}**