Yash Kandi
219303228
DSA

Assignment -2 DSA
3CCE - Sec A

Q.1 WAP to Delete Middle node of an linked list.

```
#include <stdio.h>
#include <stdlib.h>
struct node {
    int data;
    struct node *next;
} * head = NULL;
struct node *front;
void create (int key) {
    struct node * temp;
    if (head == NULL) {
        head = (struct node*) malloc (sizeof (struct node));
        head -> data = key;
        head -> next = NULL;
        front = head;
    }
    else {
        temp = (struct node*) malloc (sizeof (struct node));
        temp -> data = key;
        temp -> next = NULL;
        front -> next = temp;
        front = temp;
    }
}
```

```
void display (){
    struct node *ptr;
    ptr = head;
    if (ptr != NULL){
      while (ptr != NULL){
          printf("%d ->", ptr->data );
          ptr = ptr->next;
      } }
      else {
          printf("list is empty \n");
      }
      printf("NULL\n");
}

void deletmid (){
    struct node *ptr1;
    struct node *ptr2;
    struct node *prev;
    ptr1 = ptr2 = head;
    if (ptr1->next == NULL){
        head = NULL;
    }
    else {
    while (ptr2 != NULL && ptr2->next != NULL){
        prev = ptr1;
        ptr1 = ptr1->next;
        ptr2 = ptr2->next->next;
    }
    prev->next = ptr1->next;
  }
}
```

```
int main () {
    printf ("If one node is there :- \n");
    create (10);
    display ();
    deletemid ();
    display ();

    printF(" If there are odd number of nodes :- ");
    create (10);
    create (20);
    create (30);
    create (40);
    create (50);
    display ();
    deletemid ();
    display ();
    printf(" If there are even number of nodes :- ");
    create (60);
    create (70);
    display ();
    deletemid ();
    display ();
    return 0; }
```

Output :- If one node is there :- 10 → NULL
It is empty NULL
If there are odd number of nodes :- 10→20→30→40→50→NULL
10 → 20 → 40 → 50 → NULL
If there are even number of nodes :- 10→20→40→50→60→70→NULL
10 → 20 → 30 → 40 → 60 → 70 → NULL

Q.2 Find Intersection point of two single linked list-

```
void intersection(){
  int count=1;
  for(struct node *link1=head; link1!=NULL;
     link1=link1->next){
     count++;
     for(struct node *link2=head2; link2!=NULL;
        link2=link2->next){
        if(link1->next==link2->next){
          link1=link1->next;
          printf("%d\n", count);
          printf("Link Intersected at %d", link1->data);
          return;
        }
     }
  }
}
int main(){
  create(30); create(20); create(30); create(40);
  create(50); create(60); display(head);
  create2(101); create2(102);
  front2->next=head->next->next;
  display(head2);
  intersection(); }
```

Output :- 30 → 20 → 30 → 40 → 50 → 60 → NULL

101 → 102 → 30 → 40 → 50 → 60 → NULL

3

Link Intersection at 30

Q.3 Implement the output and iput restricted Deque

a) Input- restricted
   - enQueueRear
   - deQueueRear
   - deQueueFront

b) Output restricted
   - enQueueRear
   - enQueueFront
   - deQueueFront

Display()

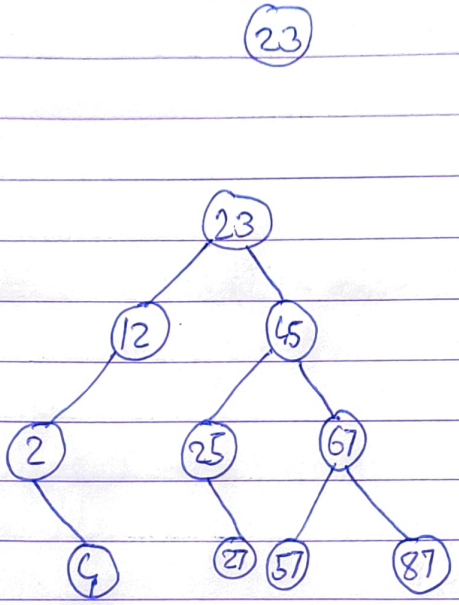
```
voidenQueueRear (int value) {
    char ch;
    if (front == SIZE/2) {
        printf("\n Queue is full! Insertion is not possible!");
        return;
    }
    do { printf("\n Enter the value to be inserted :");
        scanf("%d", &value);
        queue[front] = value;
        front++;
        printf("Do you want to continue insertion Y/N");
        ch = getch();
    }
    while (ch == 'y');
}
```

```
void enQueueFront (int value) {
    char ch;
    if (front == SIZE/2) {
        printf("\n Queue is full! Insertion is not
                possible");
        return; }
    do {
        printf("\n Enter the value to be inserted: ");
        scanf("%d", &value);
        rear--;
        queue[rear] = value;
        printf("Do you want to continue insertion Y/N");
        ch = getch();
    }
    while (ch == 'y');
}
void deQueueRear() {
    int deleted;
    if (front == rear) {
        printf("\n Queue is Empty !!! Deletion is not
                possible !!!");
        return 0;
    }
    front--;
    deleted = queue[front+1];
    return deleted;
}
```

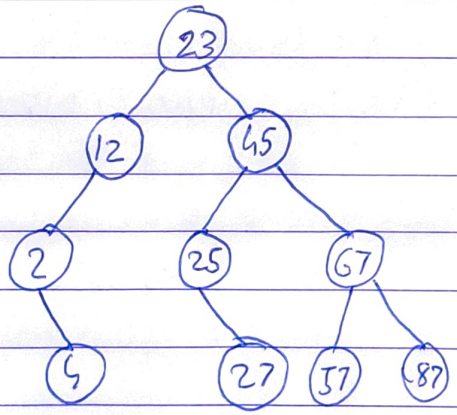
```
int dequeueFront () {
    int deleted;
    if (front == rear) {
        printf("\n Queue is Empty ! Deletion is not possible!");
        return 0;
    }
    rear ++;
    deleted = queue [rear-1];
    return deleted;
}
void display () {
    int i;
    if (front == rear)
        printf("\n Queue is Empty ! Deletion is not possible");
    else {
        printf("\n the Queue element are :");
        for ( i=rear ; i < front ; i++) {
            printf( "%d \t", queue [i]);
        }
    }
}
```

Q.4 Binary Search Tree - {23, 45, 67, 57, 87, 12, 25, 2, 4, 27}
and delete [45, 35]

Step 1 -

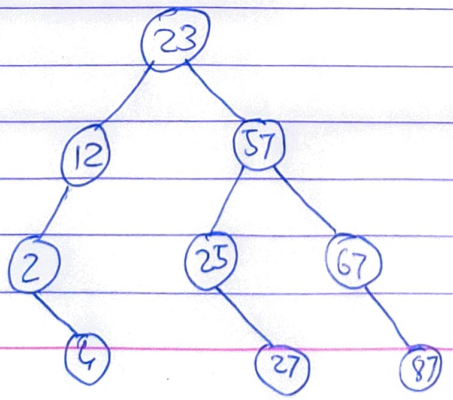



Step 2 - delete 45, 35




Now delete 45

∴



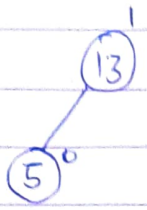
Q.5 $(A + B \wedge C) * D + E \wedge F$

| Infix | Stack | Postfix |
|---|---|---|
| A | ( | A |
| + | ( + | A |
| B | ( + | AB |
| ^ | ( + ^ | AB |
| C | ( + ^ | ABC |
| ) | ~~( + A)~~ | ABC^+ |
| * | * | ABC^+ |
| D | * | ABC^+D |
| + | + | ABC^+D* |
| E | + | ABC^+D*E |
| ^ | + ^ | ABC^+D*E |
| F | + ^ | ABC^+D*EF |
| | | ABC^+D*EF^+ |

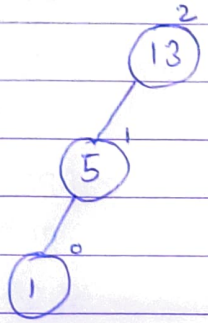
Q.8 Create AVL tree on
List data - 13, 85, 1, 7, 8, 18, 67, 26

Step 1- Insert 13

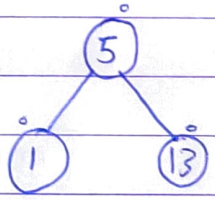
Step 2 - Insert 5



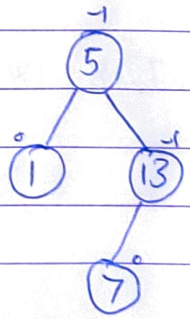

Step 3 - Insert , 1



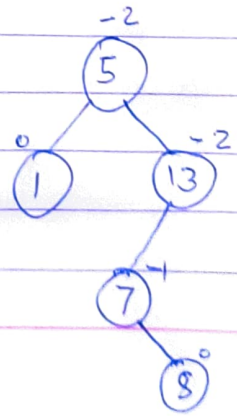

Unbalanced
Perform RR Rotation




Step 4 - Insert 7



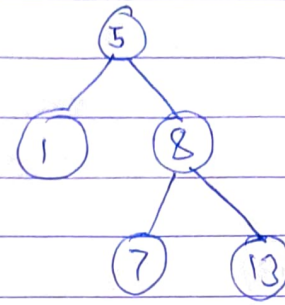

Step 5 - Insert 8



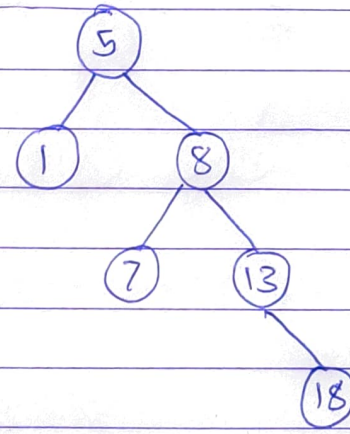

Do LR Rotation

Q.9

In - order -

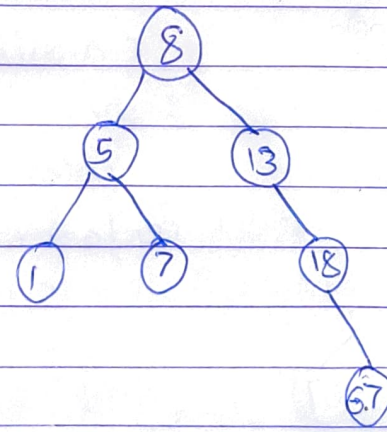



Step 6- Insert 18

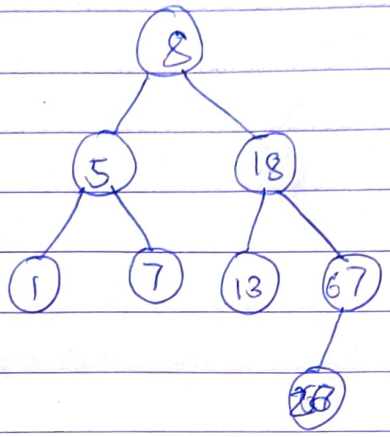



Perform
LL Rotation

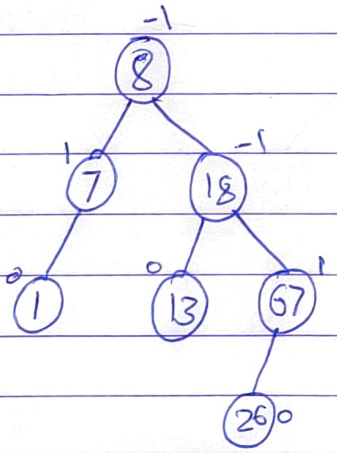
Step 7- Insert 67



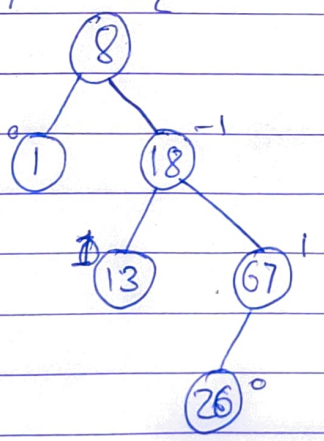
Step 8 - Insert 26



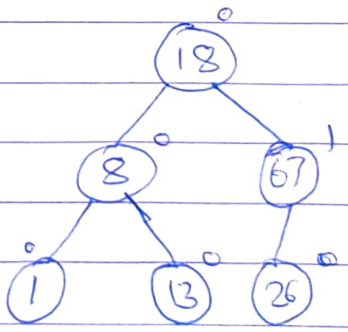

Step 9 - Delete 5



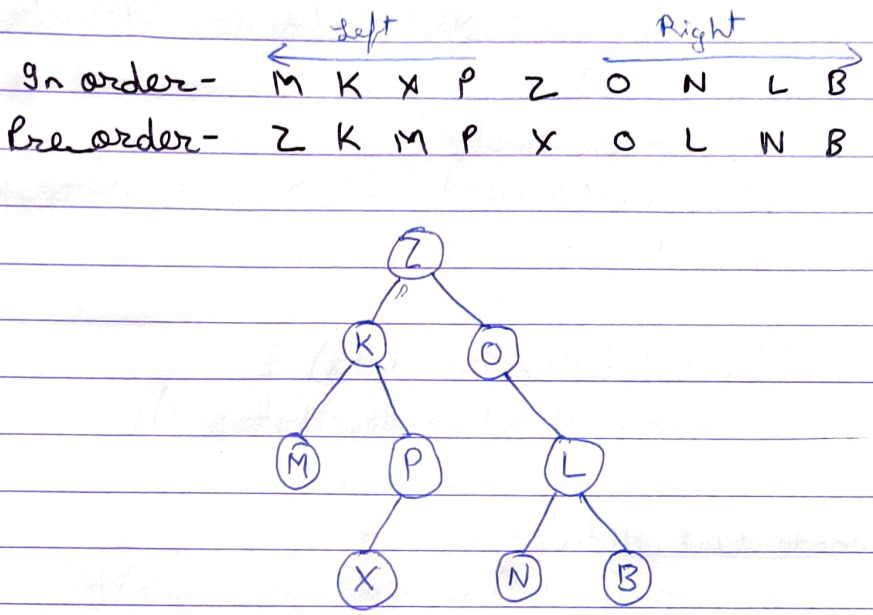

Step 10 - Delete 7




Perform LL Rotation



Q.9




Post order - L - Ri - Root

MXPKNBLOZ

Q.10

```
# include <stdio.h>
# include <stdlib.h>
 typedef struct node {
         int data;
         struct node * left;
         struct node * right;
   } node;
int getLevelutil (node * node, int data, int level) {
        if (node == NULL)
           return 0;
        if (node -> data == data)
           return level;
```

```
    int downlevel = getLevelUtil(node->left, data, level+1);
    if (downlevel != 0)
        return downlevel;
    downlevel = getLevelUtil(node->right, data, level+1);
    return downlevel;
}
int getLevel(node* node, int data) {
    return getLevelUtil(node, data, 1);
}
node* newnode(int data) {
    node* temp = (node*) malloc(sizeof(node));
    temp->data = data;
    temp->left = NULL;
    temp->right = NULL;
    return temp;
}
int main() {
    node* root;
    int x;
    root = newNode(3);
    root->left = newNode(2);
    root->right = newNode(5);
    root->left->left = newNode(1);
    root->left->right = newNode(4);
    for(x=1; x<=5; x++) {
        int level = getLevel(root, x);
        if(level)
            printf("Level of %d is %d\n", x, getLevel
                                        (root, x));
        else
            printf("%d is not present in tree\n", x); }
    getchar();
    return 0;
}
```

Output -

Level of 1 is 3
Level of 2 is 2
Level of 3 is 1
Level of 4 is 3
Level of 5 is 2

Q.11 Post order traversing using stack

→

```
void postOrderIterative (struct Node* root) {
    if (root == NULL)
        return;
    struct Stack* stack = createStack(MAX_SIZE);
    do {
        while (root) {
            if (root->right)
                push (stack, root->right);
            push (stack, root);
            root = root->left;
        }
        root = pop (stack);
        if (root->right && peek (stack) == root->right) {
            pop (stack);
            push (stack, root);
            root = root->right;
        }
        else {
            printf ("%d", root->data);
            root = NULL; }
        while (!isEmpty (stack));
    }
```

```
int main () {
    struct Node * root = NULL;
    root = newNode (1);
    root -> left = newNode (2);
    root -> right = newNode (3);
    root -> left -> left = newNode (4);
    root -> left -> right = newNode (5);
    root -> right -> left = newNode (6);
    root -> right -> right = newNode (7);
    printf ("Post order traversal of binar tree is \n");
    printf ("[");
    postorderIterative (root);
    printf ("]");
    return 0;
}
```

Output :-

Post order traversal of binary tree is:
[4 5 2 6 7 3 1]