13) WAP to implement Queues using Single Linked list

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

struct node{
  int data;
  struct node* next;
};

struct Queue {
  struct node* front;
  struct Node* rear;
};

struct Queue * initialize Queue (){
  struct Queue * queue = (struct (Queue* ) malloc (sizeof (
    struct Queue));
  return Queue;
}

int isEmpty (struct Queue* Queue) {
  return (Queue -> front == NULL);
}

void enqueue (struct Queue * queue, int newData)
{
  struct node* newNode = (struct Node*) malloc (sizeof(
  (struct node));
  newNode -> data = newData;
  newNode -> next = NULL;

  if (isEmpty(Queue)) {
    Queue -> front = Queue -> rear = newNode;
  }
  else{
    Queue -> rear -> next= newNode;
    Queue -> rear = newNode;
  }
  printf ("%d enqueued to the Queue \n", newData);
}
void dequeue (struct Queue* Queue) {
  if (isEmpty (Queue))
    {
    printf (" Queue is empty. cannot dequeue \n");
    return;
    }
  struct node * temp = Queue -> front;
  Queue -> front = temp -> next;
  if (Queue -> front == NULL)
  {
    Queue -> rear = NULL;
  }
  printf (" d dequed from Linked list Queue") temp -> data);
  free(temp);
}
```

```
void printQueue (struct
}
if (isEmpty(queue)) {
    printf("Queue is empty" \n");
    return;
}
printf("Queue Elements");
struct node * current = queue -> front;
while (current != NULL)
{
  printf("%d", current -> data);
  current = current -> next;
}
printf("\n");
int main()
{
struct Queue* queue = initializeQueue();
enqueue(queue, 3);
enqueue(queue, 4);
enqueue (queue, 5);
printQueue (queue);
dequeue(queue);
dequeue(queue);
dequeue (queue);
printQueue (queue);
return 0;
}
```

output

```
3 enqueued to the queue
4 enqueued to the queue
5 enqueued to the queue
3 dequeued from the queue
4 dequeued from the queue
5 dequeued from the queue
Queue is empty
```

14) WAP to implement Doubly linked list, insert a new node to left, delete node based on value.

```
#include<stdio.h>
#include <stdlib.h>
struct node{
   int Data;
   struct node* prev;
   struct node* next;
};
struct node* create node(int data){
 struct node * newNode = (struct Node*) malloc(sizeof(
 struct node));
 newnode -> data = data;
 newnode -> prev = NULL;
 newnode -> next = NULL;
 return newnode;
}
void insertLeft (struct node** head, struct Node* target, int data)
{
 struct node* new Node = createnode(data);
```

```
if(target -> prev != NULL)
 {
  target -> prev -> next = newNode;
  newNode -> prev = target -> prev;
 }

newNode -> next = target;
target -> prev -> next = newNode;
newNode -> prev = target -> prev;

}
newNode -> next = target;
target -> prev = newNode;
if (target == *head))
 {
   *head = newNode;
 }
}

void deleteNode (struct node ** head, int value)
{
struct Node* current = *head;
while (current != NULL)
 {
  if(current -> data == value)
  {
   if(current -> prev != NULL)
    {
      current -> prev -> next = current -> next;
    }
    if(current -> next != NULL);
      current -> next -> prev = current -> prev;
    }
   if (current == *head)
       {
         *head = current -> next;
       }
                                    -> free(current);
}
current = current -> next;
 }
printf(" Node not found \n");
 }
void displayList (struct Node* head)
 {
  printf(" Doubly Linked list ");
  while(head != NULL)
  {
   printf(" %d ", head -> data);
   head = head -> next;
  }
}

int main(){

struct Node* head = NULL;
int Nodescount;
printf(" Enter the number of nodes");
scanf("%d", &nodescount);
for(int i=0; i<nodes count ++){
}
int data;
printf(" Enter data for the node %d", i+1);
struct Node* newNode = createNode(data);
struct Node * newNode = createNode(data);
if (head == NULL)
  {
   head = newNode;
  }
   else{ InsertLeft(&head, head, data); }}
```

displaylist(head);

int valuetodelete;

printf("Enter value to be deleted");

scanf("%d", &valuetodelete);

deletenode(&head, valuetodelete);

displaylist(head);

return 0;

}

Sample Input Output

Enter the number of nodes for the doubly linked list : 3

Enter the value for the doubly linked list::

1

2

3

Double linked list : 3 2 1 NULL

Enter the value to be deleted from the list 3

2 1 NULL

26/02/24

* Program to Delete node in a BST

```
struct Treenode * minvaluenode(struct Treenode* node)
{
    struct Treenode* current = node;

    while(current && current -> left != NULL)
        current = current -> left;

    return current;
}

struct treenode * deletenode(struct Treenode * root,
int key)
{
    if(ROOT == NULL) return root;

    if (key < root -> val)
        root -> left = deleteNode(root -> left, key);

    else if (key > root -> val)
        root -> right = deletenode(root -> right, key);

    else {
        if (root -> left == NULL)
        {
```

19/2/24

Binary search tree :- Traverse using preorder / postorder / inorder, display the tree

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

struct Treenode{
    int data;
    struct Treenode * left;
    struct Treenode * Right;
};

struct Treenode * insert (struct Treenode* root, int data)
{
    if (root == NULL)
    {
        return createNode(data);
    }
    if (data < root -> data)
    {
        root -> left = insert (root -> left, data);
    }
    else if (data > root -> data)
    {
        root -> right = insert (root -> right, data);
        return root;
    }
}
```

```
void inorderTraversal (struct Treenode * root) {
    if (root != NULL) {
        inorderTraversal (root -> left);
        printf (" %d", root -> data);
        inorderTraversal (root -> right);
    }
}

void preorderTraversal (struct Treenode* root)
{
    if (root != NULL)
    {
        printf (" %d", root -> data);
        preorderTraversal (root -> left)
        preorderTraversal (root -> right)
    }
}

void postorderTraversal (struct Treenode * root)
{
    if (root != NULL)
    {
        postorderTraversal (root -> left)
        postorderTraversal (root -> right);
        printf (" %d", root -> data);
    }
}
```

```
void display (struct Treenode * root)
    {
      printf(" Inorder traversal : ");
      inordertraversal (root);
      printf ("\n");
      printf (" preorder traversal : ");
      preordertraversal (root);
      printf ("\n");
      printf (" postorder traversal ");
      postorder traversal (root);
      printf ("\n");
    }

int main ()
    {
      struct treenode* root = NULL;
      root = insert (root, 50);
      insert (root, 30);
      insert (root, 20);
      insert (root, 40);
      insert (root, 70);
      insert (root, 60);
      insert (root, 80);
      display (root);
    }
```

Leetcode problem : Find bottom left tree

```
void findbottomleft (struct treenode* node, int depth,
                     int * maxdepth, int * leftmostvalue)
{
    if (node == NULL)
    {
        return;
    }
    if (depth > * maxdepth) {
        * maxdepth = depth;
        * leftmostvalue = node -> val;
    }
    findbottomleft (node -> left , depth+1, maxdepth,
                    leftmostvalue);
    findbottomleft (node -> right , depth+1, maxDepth,
                    leftmostvalue);
}
int findbottomvalue ( struct treenode * root )
{
    int maxdepth= 0;
    int leftmostvalue = root -> val;
    findBottomleft (root, 1, &maxdepth, &leftmost
                                        value
    return leftmostvalue;}
```