```
        pop();
        break;
      case 3:
        display();
        break;
      default:
        printf("Wrong choice\n");
      }
    }
  }
}

void push(int n) {
    if (top == size-1) {
      printf("Stack Overflow condition\n");
      }
      else {
       top++;
       stack[top]=n;
       printf("PUSH() Operation is successfull\n");
      }
    }

  void pop() {
   if (top == -1) {
      printf("Stack underflow condition\n");
      }
      else {
        %d is
       printf("Stack pop() Operation successfully\n", stack[top]);
        top--;
      }
    }
    void display () {
     if (top == -1) {
          printf("Stack is empty");
```

Ended 11/2/24

Q2. WAP
in
in
int
ch
voi
voi
cha
int
voi
voi

```
while(temp != '('){
    postfix[pos] = temp;
    pos++;
    temp = pop();
  }
  break;
case '+': push()
case '-': push()
case '*': push()
case '/': push();
case '^': push()
while (pred(stack[top]) >= pred(symbol)) {
    temp = pop();
    postfix[pos++] = temp;
  }
}
void push(char symbol){
```

OUTPUT

(K+L-M*N+(O^P)*W/U/V*T+Q)

Infix expression : (K+L-M*N+(O^P)*W/U/V*T+Q)

Postfix expression : KL+MN*-OP^W*U/V/T*+Q+

3.1.2024

3.

```
        else
        printf("Item deleted = %d\n", item);
        break;
    case 3: display();
        break;
    default: exit(0);
    }
}
getch();
```

OUTPUT:-

Enter the operation

1. Insert
2. Delete
3. Display
4. Exit

1

Enter the number

5

Successfully inserted.

printf("Original Linked List 2:\n");
display(list2);

struct Node* concatenatedList = concatenateLinkedLists(list1, list2);

printf("Concatenated Linked List:\n");
display(concatenatedList);

free(list1);
free(list2);

return 0;
}

Demonstrated
Lab teacher
29/11/24

(8) Stack implementation using linked list :-

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

struct Node {
    int data;
    struct Node* next;
};
void display (struct Node* top) {
    if (top != NULL){
        printf("Stack elements are:\t");
        while (top != NULL){
            printf("%d\t", top->data);
            top = top->next;
        }
        printf("\n");
        else {
        printf ("Stack is empty\n");
    }
}
struct Node* push (struct Node * top, int data){
    struct Node* newNode = (struct Node*) malloc (sizeof (struct Node));
    if (newNode == NULL) {
        printf("Stack Overflow\n");
        return top;
    }
    newNode->data = data;
    newNode->next = top;
    top = newNode;
    return top;
}
```

SLL - Queues, Stacks, Strings
Emtd

NLP
25/1/24

Enter operation: 4
Enter position: 3
Enter operation: 2
5
2
3
9
Enter operation-1
operation completed.

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

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

void preorder (struct BST * root)
  { if (root != NULL)
      printf ("%d\t", root -> data);
      preorder (root -> left);
      preorder (root -> right);
    }
}

void main ()
{ int choice; char ch;
  printf (" Enter operation \n 1. create \n 2. display using inorder \n
      3. display using postorder \n 4. display using preorder \n
      5. -1 to end \n");
  while (1)
  {
    printf (" enter operation ");
    scanf ("%d", &choice);
    if (choice == -1)
```

BST, Lecture 12

LAB-04

1. Breadth First Search:-

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

#define MAX_NODES 100

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

struct Graph {
    int numNodes;
    struct Node* adjLists[MAX_NODES];
    int visited[MAX_NODES];
};

struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*) malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

struct Graph* createGraph(int n) {
    struct Graph* graph = (struct Graph*) malloc(sizeof(struct Graph));
    graph->numNodes = n;
    for (int i=0; i<n; i++) {
        graph->adjLists[i] = NULL;
        graph->visited[i] = 0;
    }
    return graph;
}
```

```
    newNode->next = graph->adjLists[src];
    graph->adjLists[src] = newNode;

    newNode = createNode(src);
    newNode->next = graph->adjLists[dest];
    graph->adjLists[dest] = newNode;
}

void DFS(struct Graph* graph, int startNode) {
    graph->visited[startNode] = 1;
    printf("%d ", startNode);

    struct Node* temp = graph->adjLists[startNode];
    while(temp) {
        int adjNode = temp->data;
        if(!graph->visited[adjNode]) {
            DFS(graph, adjNode);
        }
        temp = temp->next;
    }
}

int main() {
    int numNodes;
    printf("Enter the number of nodes: ");
    scanf("%d", &numNodes);

    struct Graph* graph = createGraph(numNodes);

    int numEdges;
    printf("Enter the no of edges: ");
    scanf("%d", &numEdges);

    for(int i=0; i<numEdges; i++) {
        int src, dest;
```

Lab Code :-

(a) Delete Node in a BST :-

```
struct TreeNode* inOrderSuccessor(struct TreeNode* root) {
    struct TreeNode* curr = root;
    while (curr && curr->left != NULL) curr = curr->left;
    return curr;
}

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) {
            struct TreeNode* temp = root->right;
            free(root);
            return temp;
        }
        if (root->right == NULL) {
            struct TreeNode* temp = root->left;
            free(root);
            return temp;
        }
        else {
            struct TreeNode* temp = inOrderSuccessor(root->right);
            root->val = temp->val;
            root->right = deleteNode(root->right, temp->val);
        }
    }
    return root;
}
```

26/2/24

(b) Bottom value tree :-

```
int findBottomLeftValue(struct TreeNode* root) {
    if (root == NULL) {
        return root;
    }

    struct TreeNode* queue[10000];
    int front = 0, rear = 0;
    int level_size = 0, leftmost = 0;

    queue[rear++] = root;
    while (front < rear) {
        level_size = rear - front;
        for (int i = 0; i < level_size; i++) {
            struct TreeNode* node = queue[front++];
            if (i == 0) {
                leftmost = node->val;
            }
            if (node->left) {
                queue[rear++] = node->left;
            }
            if (node->right) {
                queue[rear++] = node->right;
            }
        }
    }
    return leftmost;
}
```

26/2/24