Lab 1

18/12/25

Date / /

Page

1. Write a program to simulate the working of stack using an array with the following:
a) Push
b) Pop
c) Display

```
#include <stdio.h>
int top=-1;
void push(int arr[], int value){
if(top>4){
        printf("\nstack overflow can't insert the
                element");
  }
else{
++top;
arr[top]=value;
printf("\nSuccessfully pushed the element %d",
                arr[top]);
}
}
void pop(int arr[]){
 if(top==-1){
        printf("\n stack underflow no element
                to pop\n");
  }
else{
   printf("\none Element was poped %d\n",
                arr[top]);
      top--;
}
}
```

```
void display(int arr[]){
    printf("The elements are \n");
for (int i=top; i>=0; i--){
    printf("%d\n", arr[i]);
}
}
void main(){
    int stack[5];
    void operations(){
    int choice;
    int value;
    printf("\n Enter your choice \n");
    printf("Enter 1 to push \n Enter 2 to
    pop \n Enter 3 to display \n
        Enter 4 for Exit \n");
    scanf("%d", &choice);
    switch(choice){
case 1:
    printf("Enter the element to push");
    scanf("%d", &value);
    push(stack, value);
    operations();
    break;
case 2:
    pop(stack);
    operations();
    break;
case 3:
    display(stack);
    operations();
    break;
case 4:
    printf("You have exited");
    }
3 operations(); }
```

o/p

→ Enter your choice
Enter 1 to push
Enter 2 to pop
Enter 3 to display
Enter 4 for exit
1
Enter the Element to push 2

Successfully pushed the element 2.

→ Enter your choice
Enter 1 to push
Enter 2 to pop
Enter 3 to display
Enter 4 for exit
2.
One Element was popped 3

01/01/24

LAB-3

① Write AP to convert a given valid parenthesized infix arithmetic expression. The expression consists of single character operands and the binary operators + (plus), - (minus), * (multiply), / (divide) and ^ (power).

```
#include<stdio.h>
#include<stdlib.h>
#define MAX_SIZE 100
char stack[MAX_SIZE];
int top = -1;

void push (char item)
{
    if (top == MAX_SIZE-1) {
        printf ("Stack Overflow\n");
        exit(0);
    }
    top++;
    stack[top] = item;
}

char pop () {
    if (top == -1) {
        printf ("Stack Underflow\n");
        exit (1);
    }
    return stack[top--];
}

int isOperator (char ch) {
    return (ch == '+' || ch == '-' || ch == '*'
            || ch == '/' || ch == '^');
}

int precedence (char ch) {
    switch (ch) {
        case '^':
            return 3;
        case '*':
        case '/':
            return 2;
        case '+':
        case '-':
            return 1;
        default:
            return -1;
    }
}

void infixtoPostfix (char infix[]) {
    char postfix[MAX_SIZE];
    int i, j = 0;

    for (i = 0; infix[i] != '\0'; i++) {
        if (isalnum (infix[i])) {
            postfix[j++] = infix[i];
        }
        else if (infix[i] == '(') {
            push (infix[i]);
        }
        else if (infix[i] == ')') {
```

```
while(top != -1 && stack[top] != '('){
        postfix[j++] = pop();
    }
if (top != -1 && stack[top] == '('){
        pop();
    }
    else{
        printf("Invalid expression\n");
        exit(1);
    }
}
else if (isOperator(infix[i])){
    while(top != -1 && precendence(stack[top]
        >= precedence(infix[i]))
    {
        j++;
        postfix[j] = pop();
    }
    push(infix[i]);
}
else{
    printf("Invalid character in
                expression\n");
    exit(1);
}
}
while(top != -1){
    if(stack[top] == '('){
        printf("Invalid Expression\n");
        exit(1);
    }
    postfix[j++];
        postfix[j] = pop();
}
postfix[j] = '\0';
printf("Postfix Expression: %s\n", postfix);
}

int main(){
    char infix[max_size];
    printf("Enter a valid parenthesized infix
        arithmetic expression:");
    scanf("%s", infix);
    infixtoPostfix(infix);

    return 0;
}
```

o/p

Enter a valid parenthesized infix arithmetic expression: (a+b*c-(a+b))*a

postfix Expression: abc*+ab+-a*

(2) WAP to stimulate the working of a queue of integers using an array. Provide the following operations

a) Insert
b) Delete
c) Display

```
#include <stdio.h>
#define max_size 10

void insert (int queue[], int *rear, int item);
void delete (int queue[], int *front, int rear);
void display (int queue[], int front, int rear);

int main(){
    int queue[max_size];
    int front = -1, rear = -1;
    int choice, item;

    do{
        printf("\n1. Insert\n2. Dele\n
            3. Display\n");
        printf("Enter your choice:");
        scanf("%d", &choice);
        switch(choice){
            case 1:
                printf("Enter element to insert:");
                scanf("%d", &item);
                insert(queue, &rear, item);
                break;
            case 2:
                delete(queue, &front, rear);
                break;
            case 3:
                display(queue, front, rear);
                break;
            case 4:
            exit:
            default:
                printf("Invalid choice!");
        }
    } while (choice != 4);
    return 0;
}

void insert (int queue[], int *rear, int item)
{
    if(*rear == max_size - 1){
        printf("Queue Overflow ! Cannot insert.\n");
        return;
    }
    if (*rear == -1){
        *rear = 0;
        queue[*rear] = item;
        printf("Element %d inserted into
            the queue.\n", item);
    }
    else{
        (*rear)++;
        queue[*rear] = item;
        printf("Element %d inserted into the
            queue.\n", item);
    }
}
```

```
void delete (int queue[], int *front, int rear)
{
    if (*front == rear) {
        printf("Queue is empty");
        return;
    }
    printf("Element %d deleted from the
            queue.\n", queue[++(*front)]);
}

void display (int queue[], int front, int rear)
{
    if (front == rear) {
        printf ("Queue is empty.\n");
        return;
    }
    printf ("Queue elements:");
    for (int i = front + 1; i <= rear; i++) {
        printf ("%d", queue[i]);
    }
    printf("\n");
}
```

o/p

-> 1. Insert 2. Delete 3. Display
Enter your choice: 1
Enter the element to Insert: 2
Element 2 Inserted into the queue

-> 1. Insert 2. Delete 3. Display
Enter your choice: 1
Enter the element to Insert: 3
Element 3 Inserted into the queue.

-> 1. Insert 2. Delete 3. Display
Enter your choice: 3
Queue Elements: 2 3

-> 1. Insert 2. Delete 3. Display
Enter your choice: 2
Element 2 deleted from the queue.

// WAP to Stimulate the working of circular Queue

```
#include <stdio.h>
# define N 5;

int queue[N];
int front = -1;
int rear = -1;

void enqueue(int x){
    if (front == 0 && rear == max-N-1)
    {
        printf("overflow!");
    }
    else if{
        if (front == -1 && rear == -1)
        {
            front = rear = 0; queue[rear] = x;
        }
    else if (Rear = Max-1 and front != 0
        {
            rear = 0; queue[rear] = x;
        }
    else
        {
            rear++;
            queue[rear] = x;
        printf("%d is in queue", x);
    }
}
```

```
void dequeue(){ int x;
    if (front == -1)
    {
        printf("Underflow");
    }
    else if (front = rear)
    {
        front = rear = -1;
    }
    else if (front == N-1){
        front = 0
        queue[front] = x;
    }
    else{
        front++;
        x = queue[front];
        printf("%d is removed", x);
    }
void display(){
    if (front == -1 && rear == -1)
    {
        printf("Underflow");
    }
    else
    {
        for(i=front; i<=rear+1; i++){
        {
            printf("%d ", queue[i];
        }
    }
}
```

```
void main()
{
    int item;
    int choice;
    do{
        printf("1 for enqueue\n 2 for
          dequeue \n 3 for display\n");
        scanf("%d",&choice);
        switch(choice){
            case 1:
                printf("Enter data:");
                scanf("%d",&item);
                enqueue(item);
                break;
            case 2:
                dequeue();
                break;
            case 3:
                display();
                break;
        }
    }while(choice!=0);
}
```

dp

```
1 for enqueue
2 for dequeue
3 for display

1
Enter data:
1
1 is in the queue

1 for enqueue
2 for dequeue
3 for display
1

Enter data: 2
2 is in the queue

1 for enqueue
2 for dequeue
3 for display
2

1 is removed from the queue

1 for enqueue
2 for dequeue
3 for display
1

Enter data: 6
6 is in the queue
```

8/1/2019

08/01/2024

Lab-4 (Leet Code)

Find first and last position of element in sorted array

```
#include<stdio.h>
#include<stdlib.h>
int binarySearch(int * nums, int numSize,
        int target, int findFirst){
    int left=0;
    int right = numSize-1;
    int result=-1;
    while (left<=right){
        int mid=left+(right-left)/2
        if (nums[mid]==target){
            result=mid;
            if (findFirst)
                right=mid-1
            else
                left=mid+1;
        }
        else if (nums[mid]<target){
            left=mid+1;
        }
        else{
            right=mid-1;
        }
    }
    return result;
}
```

```
int * searchRange(int * nums, int numsSize,
        int * returnSize){
    int * result = (int*)malloc(2 * sizeof(int));
    *returnSize = 2;
    int firstOccurence = binarySearch(nums,numsSize,
        target,1);
    int lastOccurence = binarySearch(nums, numsSize,
        target,0);
    return
    result[0] = firstOccurence;
    result[1] = lastOccurence;

    return result;
}

int main(){
    int nums[] = {5,7,7,8,8,10};
    int target = 8;
    int numsize = sizeof(nums)/sizeof(nums[0]);

    int returnSize;
    int * result = searchRange(nums,
        numsSize, target, &returnSize);
    printf("starting position: %d\n", result[0]);
    printf("Ending position: %d\n", result[1]);

    free(result);

    return 0;
}
```

o/p

nums = [5,7,7,8,8,10]

target = 8

[3,4]

Lab-5

Implementation of Linked List
* Insertion At End
* Inserting At Beginning
* Insertion At Given Position

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

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

void insert_at_beg (struct node ** head, int data)
    struct node * new_node = (struct node*)
        malloc (sizeof (struct node));
    new_node -> data = data;
    if (*head == NULL) {
        *head = new_node;
        new_node -> next = NULL;
    } else {
        new_node -> next = *head;
        *head = new_node;
    }
}
void insert_in_between (struct node ** head, int da
{
    int pos;
    printf("Enter the position where to insert
        data : \n");
    scanf ("%d", &pos);
    int count = 1;
    struct node * ptr = *head;
    if (*head == NULL) {
        printf("No Nodes, can't insert @
            position");
    }
    while (ptr -> next != NULL) {
        if (count == (pos-1)) {
            struct node * new_node;
            new_node = malloc (sizeof (struct node));
            new_node -> data = data;
            new_node -> next = ptr -> next
            ptr -> next = new_node;
            return;
        }
        count ++;
        ptr = ptr -> next;
        if (ptr -> next == NULL) {
            printf("Reached the end node can't
                insert at specified node \n");
            return;
        }
    }
}
void insert_at_end (struct node ** head, int data) {
    struct node * new_node = malloc (sizeof (struct node));
    new_node -> data = data;
    new_node -> next = NULL;
    if (*head == NULL) {
        *head = new_node;
        return;
    }
```

```
struct node * ptr = *head;
while (ptr -> next != NULL) {
    ptr = ptr -> next;
}
ptr -> next = new_node;
}
void display (struct node ** head) {
    if (*head == NULL) {
        printf("No Nodes \n");
        return;
    }
    struct node * ptr = *head;
    while (ptr != NULL) {
        printf("Data = %d \n", ptr -> data);
        ptr = ptr -> next;
    }
}

int main() {
    struct node * head = NULL;
    insert_at_beg (&head, 10);
    insert_at_end (&head, 20);
    insert_at_beg (&head, 30);
    insert_at_end (&head, 40);
    printf("Before Inserting : \n");
    display (&head);
    printf("After Inserting 50 @ the
        Beginning : \n");
    insert_at_beg (&head, 50);
    display (&head);
    printf("After Inserting 80 @ the end : \n");
    insert_at_end (&head, 80);
    display (&head);
    return 0;
}
```

O/P

Before Inserting :

Data = 30
Data = 10
Data = 20
Data = 40

After Inserting 50 at beginning :
Data = 50
Data = 30
Data = 10
Data = 20
Data = 40

After Inserting 80 at end :
Data = 50
Data = 30
Data = 10
Data = 20
Data = 40
Data = 80

Enter the pos where to Insert data : 3
After Inserting 100 @ position 3 :
Data = 50
Data = 30
Data = 100
Data = 10
Data = 20
Data = 40
Data = 80

Implementation of linked list

1. Deleting @ beginning
2. Deleting @ end
3. Deleting @ given position.

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

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

void insert_at_beg(struct node ** head,
                   int data){
  struct node * new_node = (struct node *)
      malloc(sizeof(struct node));
  new_node -> data = data;
  if(*head == NULL){
      *head = new_node;
      new_node -> next = NULL;
  }
  else{
      new_node -> next = *head;
      *head = new_node;
  }
}
```

```
void display(struct node ** head){
    if(*head == NULL){
        printf("No nodes\n");
        return;
    }
    struct node * ptr = *head;
    while(ptr != NULL){
        printf("Data = %d\n", ptr->data);
        ptr = ptr->next;
    }
}
void delete_at_beg(struct node ** head){
    if(*head == NULL){
        printf("Underflow\n");
    }
    else{
        struct node * temp = *head;
        *head = temp->next;
        free(temp);
    }
}
void delete_at_end(struct node ** head){
    if(*head == NULL){
        printf("Underflow\n");
        return;
    }
    struct node * ptr = *head;
    struct node * prev;
    if(ptr->next == NULL){
        *head = NULL;
        free(ptr);
        return;
}
```

```
while (ptr -> next != NULL){
    prev = ptr;
    ptr = ptr -> next;
}
prev -> next = NULL;
free (ptr);
}

void delete_element (struct node ** head){
    int element;
    printf ("Enter the Element to Delete\n");
    scanf ("%d", &element);
    if (*head == NULL) {
        printf ("underflow");
        return;
    }
    struct node * ptr = *head;
    struct node* prev;
    if (ptr -> next == NULL) {
        if (ptr -> data == element) {
            free (ptr);
            *head = NULL;
        }
    }
    else{
        while (ptr != NULL) {
            if (ptr -> data == element) {
                if (ptr -> next == NULL) {
                    prev -> next = NULL;
                    free (ptr);
                    return;
                }
                else {
                    prev -> next = ptr -> next;
                    free (ptr);
                    return;
                }
            }
            prev = ptr;
            ptr = ptr -> next;
            if (ptr == NULL) {
                printf ("Element not found \n");
            }
        }
    }
}
int main() {
    struct node * head = NULL;
    printf ("Before deleting :\n");
    insert_at_beg (&head, 10);
    insert_at_beg (&head, 20);
    insert_at_beg (&head, 30);
    display (&head);
    printf ("After Deleting an element at beg:");
    delete_at_beg (&head);
    printf ("After deleting an element @ end:");
    delete_at_end (&head);
    display (&head);
    printf ("After deleting an element at whose
                data = 30:\n");
    delete_element (&head);
    display (&head);
    return 0;
}
```

o/p

Before Deleting:
Data = 40
Data = 20
Data = 10
Data = 5
After Deleting element at beginning:

Data = 20
Data = 10
Data = 5
After Deleting element at end;

Data = 20
Data = 10

After Deleting element at where data = 20.

Data = 10

22/01/24

pgm4.c

Lab-6
(Stack - Linked - List)

```
#include<stdio.h>

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

void push (struct node ** head, int data ) {
    struct node * new_node = (struct node *)
      malloc (sizeof (struct node));
    new_node -> data = data;
    if (*head == NULL) {
       *head = new_node;
       new_node -> next = NULL;
    }
    else {
       new_node -> next = *head;
       *head = new_node;
    }
}

void pop (struct node ** head ) {
    if (*head == NULL) {
       printf("Underflow\n");
    }
    else {
       struct node * ptr = *head;
       *head = ptr -> next;
       free(ptr);
    }
}
```

```
void display (struct node ** head) {
    struct node * ptr = * head;
    while (ptr != NULL) {
        printf ("Data = %d \n", ptr -> data);
        ptr = ptr -> next;
    }
}
int main () {
    struct node * head = NULL;
    push (& head, 20);
    push (& head, 40);
    push (& head, 60);
    display (& head);
    printf (" After popping \n ");
    pop (& head);
    display (& head);
    return 0;
}
```

Queue-Linked-List

```
#include <stdio.h>

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

void enqueue (struct node ** head, struct node
        ** tail, int data) {
    struct node * new_node = malloc (sizeof (struct node));

    new_node -> data = data;
    if (*head == NULL) {
        * head = new_node;
        * tail = new_node;
        new_node -> next = NULL;
    }
    else {
        struct node * ptr = * tail;
        ptr -> next = new_node;
        * tail = new_node;
        new_node -> next = NULL;
    }
}
void dequeue (struct node ** head, struct node
        ** tail) {
    if (head == NULL) {
        printf ("Underflow \n");
    }
    else if (* head == * tail) {
        * head = NULL;
        * tail = NULL;
    }
```

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

int main () {
    struct node * head = NULL;
    struct node * tail = NULL;
    enqueue (&head, &tail, 20);
    enqueue (&head, &tail, 30);
    enqueue (&head, &tail, 40);
    enqueue (&head, &tail, 50);
    display (&head);
    printf ("After Deleting \n");
    dequeue (&head, &tail);
    display (&head);
    return 0;
}
```

o/p

Leet code

```
int get length (struct ListNode *head) {
    int length = 0;
    while (head != NULL) {
        length ++;
        head = head -> next;
    }
    return length;
}

struct ListNode * advancelist (struct ListNode * head, int steps) {
    while (steps > 0) {
        head = head -> next
        steps --;
    }
    return head;
}
struct ~~ListNode~~ ListNode
    * getIntersectionNode (struct ListNode*
        head A, struct ListNode *head
{
    int length A = get length (head A);
    int length B = get length (head B);
    if (length A > length B) {
        head A = advancelist (head A,
            length A - length B);
    }
    else {
        head = advancelist (head B, length B -
            length A);
    }
```

```
while (headA != NULL && headB != NULL)
    if (head == headB) {
        return headA;
    }
    headA = headA -> next;
    headB = headB -> next;
}
return NULL;
}
struct ListNode *CreateNode (int val) {
    struct ListNode *newnode =
        (struct ListNode *) malloc(
        sizeof (struct ListNode));
    if (newNode == NULL) {
        printf ("memory allocation failed\n");
        exit (1);
    }
    newNode -> val = val;
    newnode -> next = NULL;
    return NewNode;
}
void freeLinkedList (struct ListNode *head);
    struct ListNode *temp;
    while (head != NULL) {
        temp = head;
        head = head -> next
        free (temp);
    }
}
```

Q/r

Case 1	Case 2	Case 3
[4, 1, 8, 4, 5]	2	0
[5, 6, 1, 8, 4, 5]	[1, 9, 1, 2, 4]	[2, 6, 4]
2	[3, 2, 4]	[1, 5]
3	3	3
	1	2

Intersected at 8 | Intersected at 2 | No Intersection

05/02/24

Lab - 7

* Reversing a Linked list
* Concatenating a Linked list
* Sorting a Linked list.

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

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

void reverse (struct node ** head) {
    struct node * nextnode;
    struct node * prev = NULL;
    struct node * ptr = * head;
    while (ptr != NULL) {
        nextNode = ptr -> next;
        ptr -> next = prev;
        prev = ptr
        ptr = nextNode;
    }
    * head = prev;
}

void concat (struct node ** head1, struct
        node ** head2) {
    struct node * ptr = * head;
    while (ptr -> next != NULL) {
        ptr = ptr -> next;
    }
    ptr -> next = * head2;
}

void sort (struct node ** head) {
    struct node * current = * head;
    int temp;
    while (current != NULL) {
        struct node * next = current -> next;
        while (next != NULL) {
            if (current -> data >= next -> data) {
                temp = current -> data;
                current -> data = next -> data;
                next -> data = temp;
            }
            next = next -> next;
        }
        current = current -> next;
    }
}

void insert (struct node ** head, int data) {
    struct node * newNode = (struct node *)
        malloc (sizeof (struct node));
    newNode -> data = data;
    if (*head == NULL) {
        * head = newNode;
        newNode -> next = NULL;
    }
    else {
        newNode -> next = * head;
        * head = newNode;
    }
}

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

```
int main(){
    struct node * head1 = NULL;
    insert(&head1, 50);
    insert(&head1, 50);
    insert(&head1, 30);
    insert(&head1, 40);
    display(&head1);
    printf("After reversing \n");
    reverse(&head1);
    display(&head1);

    struct node * head2 = null;
    insert(&head2, 80);
    insert(&head2, 96);
    insert(&head2, 58);
    insert(&head2, 20);
    printf("Linked List 1: \n");
    display(&head1);
    printf("Linked list 2: \n");
    display(&head2);
    printf("After concatenating list\n");
    concat(&head1, &head2);
    display(&head1);
    printf("Sorting list 2\n");
    sort(&head2);
    display(&head2);
    return 0;
}
```

o/p

Data = 300
1000
30
50
80

After reversing
Data: 80
50
30
100
300

Linked List 1:
Data: 80
50
30
100
300

Linked List 2:
Data: 80
58
96
36.

After concatenating List:
Data: 80
50
30
1000
300
80
58
96
36.

Sorting list 2:
20
36
58
96

// Doubly Linked list
* Insert At Left
* deleteNode
* displayList

```
#include <stdio.h>
#include <stdlib.h>
struct node {
    int data;
    struct node * next;
    struct node * prev;
};
void insert (struct node ** head, int value) {
    struct node * newnode = malloc (sizeof (struct
                                    node));
    new_node -> data = value;
    new_node -> prev = null;
    new_node -> next = NULL;
    if (* head == NULL) {
        * head = newnode;
        return;
    }
    (*head) -> prev = newNode;
    newNode -> next = * head;
    * head = newode;
}

void deleteNode (Struct Node ** head, int key) {
    Struct node * temp = * head;
    while (temp != NULL && temp -> data != key)
        temp = temp -> next;
    if (temp == NULL) {
        return;
    }
    if (temp -> prev != NULL) {
        temp -> prev -> next = temp -> next;
    }
    if (temp -> next != NULL) {
        temp -> next -> prev = temp -> prev;
    }
    if (temp == * head) {
        * head = temp -> next;
        free (temp);
    }
}
void display (struct node * head) {
    struct node * temp = head;
    while (temp != NULL) {
        printf ("%d->", temp -> data);
        temp = temp -> next; }
    }
    printf ("NULL\n");
}
int main () {
    struct node * head = NULL;
    int data;
    int pos;
    int choice;
    do {
        printf ("\t\t\n 1 for insert at
            left of node \n 2 for delete or
            given position \n 3 for display: \n");
        scanf ("%d", & choice);
    switch (choice) {
        case 1:
            printf ("\n Enter data:");
            scanf ("%d", & data);
            insert (& head, data);
            break;
```

```
case 2:
    printf("Enter the key:");
    scanf("%d", pos);
    deleteNode(&head, pos);
    break;
case 3:
    display(head)
    break;
}
}while(choice != 0);
}
```

O/P

Enter
1 for insert at left of node
2 for delete at given pos
3 for display
1
1
2
3
2 → 1 → NULL

19/12/23

Lab-8

WAP → To construct a binary search tree
To traverse the tree using all the methods
i.e., in-order - preorder - & post order
To display the elements in the tree

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

struct node {
    int data;
    struct node *left;
    struct node *right;
};

struct node *createNode(int data){
    struct node *temp = malloc(sizeof(struct node));
    temp->left = NULL;
    temp->right = NULL;
    temp->data = data;
    return temp;
};

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

```
    return root;
};

void inorder(struct node * root) {
    if (root != NULL){
        inorder(root->left);
        printf("%d", root->data);
        inorder(root->right);
    }
}
void preorder(struct node * root){
    if (root != NULL) {
        postorder(root->left);
        postorder(root->right);
        printf("%d", root->data);
    }
}
int main(){
    struct node * root = NULL;
    root = insert(root, 20);
    insert(root, 10);
    insert(root, 30);
    insert(root, 40);
    insert(root, 70);
    insert(root, 80);
    inorder(root);
    printf("\n");
    preorder(root);
    printf("\n");
    postorder(root);
    return 0;
}
```

```
o/p
10 20 30 40 70 80
20 10 30 40 70 80
10 80 70 40 30 20
```

// Hackerrank

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

bool compare_Lists(SinglyLinkedListNode * head1,
                   SinglyLinkedListNode * head2) {
    while (head1 != NULL && head2 != NULL){
        if(head1->data != head2->data){
            return 0;
        }
        head1 = head1->next;
        head2 = head2->next;
    }
    if(head1==NULL && head2==NULL){
        return 1;
    }
    return 0;
}
```

o/p

	My Output:	Expected o/p
2	0	0
2	1	1
1		
2		
1		
1		
2		
2		
1		
2		

// LeetCode -

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

int leftheight(struct TreeNode * node){
    int height = 0;
    while(node){
        height++;
        node = node -> left;
    }
    return height;
}
int countNodes(struct TreeNode * root){
    if(!root){
        return 0;
    }
    int leftHeightval = leftHeight(root);
    int rightHeightval = leftHeight(root->right);
    if(leftHeightval == rightHeightval){
        return (1 << leftHeightval) - 1;
    }
    else{
        return 1 + countNodes(root->left) +
                   countNodes(root->right);
    }
}
```

o/p

case 1: [1 2 3 4 5 6] → 6

case 2: [] → 0

case 3: [1] → 1

26/02/24 Lab - 09

// Write a program to traverse a graph using BFS method

```
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#define MAX_VERTICES 50

typedef struct Graph{
    int V;
    bool adj[MAXVERTICES][MAX_VERTICES];
} Graph;

Graph * Graph_create(int V)
{
    Graph* g = malloc(sizeof(Graph));
    g->V = V;

    for(int i=0; i<V; i++)
    {
        for(int j=0; j<V; j++)
        {
            g->adj[i][j] = false;
        }
    }
    return g;
}
void Graph_destroy(Graph * g)
{
    free(g);
}
```

```
void Graph_addEdge(Graph * g, int v, int w)
{
    g->adj[v][w] = true;
}

void Graph_BFS(Graph* g, int s)
{
    bool visited[MAX_VERTICES];
    for(int i=0; i<g->V; i++)
    {
        visited[i] = false;
    }
    int queue[MAX_VERTICES];
    int front = 0, rear = 0;
    visited[s] = true;
    queue[rear++] = s;

    while(front != rear)
    {
      s = queue[front++];
      printf("%d", s);

      for(int adjacent = 0; adjacent < g->V;
                  adjacent++)
      {
        if(g->adj[s][adjacent] && !visited
                  [adjacent])
        {
          visited[adjacent] = true;
          queue[rear++] = adjacent;
        }
      }
    }
}
```

```
int main()
{
    Graph * g = Graph_create(4);
    Graph_addEdge(g, 0, 1);
    Graph_addEdge(g, 0, 2);
    Graph_addEdge(g, 1, 2);
    Graph_addEdge(g, 2, 0);
    Graph_addEdge(g, 2, 3);
    Graph_addEdge(g, 3, 3);

    printf("Following is Breadth first
      Traversal (starting from vertex 2)\n");
    Graph_BFS(g, 2);
    Graph_destroy(g);

    return 0;
}
```

// write a program to check whether given graph is connected or not using DFS method.

```
#include<stdio.h>
int a[20][20], reach[20], n;
void dfs(int v){
    int i;
    reach[v] = 1;
    for(i=1; i<=n; i++){
        printf("\n %d->%d", v, i);
        dfs(i);
    }
}
```

```
int main(){
    int i, j, count=0;
    printf("\n Enter number of vertices:");
    scanf("%d", &n);
    for(i=1; i<=n; i++){
        reach[i]=0;
        for(j=1; j<=n; j++)
            a[i][j]=0;
    }
    printf("\n Enter the adjacency matrix:\n");
    for(i=1; i<=n; i++)
        for(j=1; j<=n; j++)
            scanf("%d", &a[i][j]);
    dfs(1);
    printf("\n");
    for(i=1; i<=n; i++){
        if(reach[i])
            count++;
    }
    if(count==n)
        printf("\n Graph is connected");
    else
        printf("\n Graph is not connected");
    return 0;
}
```