## DS PRACTICAL SLIPS/slip1.1.c

```c
#include<stdio.h>

int binarySearch(int *arr, int n, int num){
    int low = 0;
    int high = n - 1;
    while(low <= high){
        int mid = (low + high)/2;

        if(arr[mid] == num){
            return mid;
        }

        else if(arr[mid] < num){
            low = mid + 1;
        }
        else{
            high = mid - 1;
        }
    }
    return -1;
}

int main(){
    int n;
    int num;
    printf("Enter n: ");
    scanf("%d", &n);
    int arr[n];
    printf("Enter %d elements: ", n);
    for(int i = 0; i < n; i++){
        scanf("%d", &arr[i]);
    }
    printf("Enter number to search: ");
    scanf("%d", &num);

    int res = binarySearch(arr, n, num);

    if(res != -1){
        printf("%d found at [%d] position\n", num, res);
    }
    else{
        printf("Not in array\n");
    }

    return 0;
}

```

## DS PRACTICAL SLIPS/slip1.2.c

```c
#include<stdio.h>
#include<stdbool.h>
#define MAX 10

int stack[MAX];
int top = -1;

bool isEmpty(){
    return top == -1;
}

```

```c
void push(int data){
    if(top >= MAX - 1){
        printf("Stack Overflow |||\\\n");
    }
    else{
        stack[++top] = data;
        printf("PUSHED *thumbs up*\n");
    }
}

int pop(){
    if(isEmpty()){
        printf("Stack underflow\\\\\\|\n");
        return -1;
    }
    else{
        int popped_value = stack[top--];
        return popped_value;
    }
}

void display(){
    if(isEmpty()){
        printf("Rikama aahe stack\n");
        return;
    }
    else{
        printf("Printing stack from bottom\n");
        for(int i = top; i >= 0; i--){
            printf("[%d]\n", stack[i]);
        }
    }
}

int main(){
    int choice;
    int value;

    while(1){
        printf("1. Push\n");
        printf("2. Pop\n");
        printf("3. isEmpty ?\n");
        printf("4. Display\n");
        printf("Enter choice: ");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            printf("Enter value to push: ");
            scanf("%d", &value);
            push(value);
            break;

        case 2:
            value = pop();
            if(value != -1){
                printf("Popped value is: %d\n", value);
            }
            break;

        case 3:
            if(isEmpty()){
                printf("EMPTY\n");
            }
            else{
                printf("Not Empty\n");
            }
```

```
            break;

        case 4:
            display();
            break;

        default:
            printf("Psch!! Disat nahi ka ?\n");
            break;
        }
    }
}

```

---

## DS PRACTICAL SLIPS/slip2.1.c

```c
#include<stdio.h>

void display(int arr[],int n)
{
    for( int i=0 ;i<n;i++)
    {
        printf("%d\t", arr[i]);
    }
    printf("\n");
}

void bubblySilk(int *arr, int n){
    for(int i = 0; i < n - 1; i++){
        for(int j = 0; j < n - i - 1; j++){
            if(arr[j] > arr[j + 1]){
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = arr[j];
            }
        }
    }
}

int main()
{
    int n;
    printf("Enter the element ");
    scanf("%d",&n);
    int arr[n];
    printf("Enter %d elements: ", n);
    for(int i = 0; i < n; i++){
        scanf("%d", &arr[i]);
    }
    display(arr, n);
    bubblySilk(arr, n);
    display(arr, n);

    return 0;
}

```

---

## DS PRACTICAL SLIPS/slip2.2.c

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

struct node{
    int data;
```

```c
    struct node *left;
    struct node *right;
};

struct node *create(int data){
    struct node *newnode = (struct node *)malloc(sizeof(struct node));
    newnode->data = data;
    newnode->left = newnode->right = NULL;

    return newnode;
}

struct node *insert(struct node *root, int data){
    if(root == NULL){
        return create(data);
    }

    if(data > root -> data){
        root->right = insert(root->right, data);
    }
    else if(data < root -> data){
        root->left = insert(root->left, data);
    }

    return root;
}

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

int main(){
    int n, c;
    struct node *root = NULL;
    printf("How many nodes: ");
    scanf("%d", &c);

    for(int i = 0; i < c; i++){
        printf("Enter data for node %d: ", i + 1);
        scanf("%d", &n);
        root = insert(root, n);
    }

    inorder(root);

    return 0;
}

```

---

## DS PRACTICAL SLIPS/slip3.1.c

```c
#include<stdio.h>

void display(int *arr, int n){
    for(int i = 0; i < n; i++){
        printf("[%d]\t", arr[i]);
    }
    printf("\n");
}

void insertionSort(int *arr, int n){
```

```c
    int i, j, key;
    for(int i = 1; i < n;  i++){
        key = arr[i];
        j = i - 1;
        while(j >= 0 && arr[j] >= key){
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;
    }
}

int main(){
    int n;
    printf("Enter n: ");
    scanf("%d", &n);
    int arr[n];
    printf("Enter %d elements: ", n);
    for (int i = 0; i < n; i++)
    {
        scanf("%d", &arr[i]);
    }

    printf("\n--BEFORE SORT--\n");
    display(arr, n);
    insertionSort(arr, n);
    printf("\n--AFTER SORT--\n");
    display(arr, n);

    return 0;
}

```

---

**DS PRACTICAL SLIPS/slip3.2.c**

```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<ctype.h>
#define MAX 100

int top = -1;
char stack[MAX];

void push(char item){
    if(top >= MAX - 1){
        printf("Stack overflow\n");
    }
    stack[++top] = item;
}

char pop(){
    if(top < 0){
        return '\0';
    }
    return stack[top--];
}

char peek(){
    if(top < 0){
        return '\0';
    }

    return stack[top];
}
```

```c

int isEmpty(){
    return top == -1;
}

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

int precedence(char op){
    if(op == '*' || op == '/')
        return 2;
    if(op == '+' || op == '-')
        return 1;
}

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

    for(i = 0; infix[i] != '\0'; i++){
        next_char = infix[i];

        if(isalnum(next_char)){
            postfix[j++] = next_char;
        }
        else if(next_char == '('){
            push(next_char);
        }
        else if(next_char == ')'){
            while (!isEmpty() && peek() != '('
            {
                postfix[j++] = pop();
            }
            if(peek() == '('){
                pop();
            }
        }
        else if(isOp(next_char)){
            while (!isEmpty() && precedence(peek()) >= precedence(next_char)){
                postfix[j++] = pop();
            }
            push(next_char);
        }
    }

    while (!isEmpty())
    {
        postfix[j++] = pop();
    }


    postfix[j] = '\0';

}

int main(){
    char infix[MAX];
    char postfix[MAX];

    printf("Enter Infix expression: ");
    if (fgets(infix, MAX, stdin) == NULL) {
    fprintf(stderr, "Error reading input.\n");
    return 1;
}

```

```c
    // Remove the newline character added by fgets(), if present
    infix[strcspn(infix, "\n")] = 0;

    // Convert and display the result
    infixTopostfix(infix, postfix);

        infixTopostfix(infix, postfix);
        printf("Infix Equation: %s\n", infix);
        printf("Postfix Equation: %s\n\n", postfix);

        return 0;
    }

```

---

## DS PRACTICAL SLIPS/slip4.1.c

```c
#include<stdio.h>

void linearSearch(int *arr, int n){
    int key;
    int pos;
    int flag = 0;
    printf("Enter element to search: ");
    scanf("%d", &key);

    for(int i = 0; i < n; i++){
        if(key == arr[i]){
            pos = i;
            flag = 1;
            break;
        }
    }

    if(flag == 1){
        printf("%d basho de sore o mitsukematisha: [%d]\n", key, pos + 1);
    }
    else{
        printf("%d hairetsu-nai ni arimasen\n");
    }

}

int main(){
    int n;
    printf("Enter n: ");
    scanf("%d", &n);

    int arr[n];

    printf("Enter %d elements: ", n);

    for(int i = 0; i < n; i++){
        scanf("%d", &arr[i]);
    }

    linearSearch(arr, n);

    return 0;
}

```

---

## DS PRACTICAL SLIPS/slip4.2.c

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

```c
#include<stdlib.h>

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


struct node *create(struct node *head){
    int n, data;
    struct node *temp = head;

    printf("How many nodes ");
    scanf("%d", &n);

    for(int i = 0; i < n; i++){
        struct node *newnode = (struct node *)malloc(sizeof(struct node));
        printf("Enter data for node [%d]: ", i + 1);
        scanf("%d", &data);

        newnode -> data = data;
        newnode -> next = NULL;

        if(head == NULL){
            head = newnode;
            temp = newnode;
        }
        else{
            temp->next = newnode;
            temp = newnode;
            newnode->next = head;
        }
    }
    printf("Creation successful\n");

    return head;
}

struct node *insert(struct node *head){
    struct node *newnode = NULL;
    struct node *temp = head;
    newnode = (struct node *)malloc(sizeof(struct node));
    newnode -> next = NULL;
    printf("Enter data: ");
    scanf("%d", &newnode->data);

    if(head == NULL){
        head = newnode;
        newnode->next = head;
    }
    else{
        while(temp -> next != head){
            temp = temp -> next;
        }
        newnode -> next = head;
        temp -> next = newnode;
        head = newnode;
    }
    printf("Insertion at the front successfull\n");
    return head;
}

struct node *delete(struct node *head){
    struct node *temp = head;
    struct node *temp1 = head;

```

```c
    if(head->next == head){
        free(head);
        return NULL;
    }
    else{
        while(temp->next != head){
            temp = temp -> next;
        }
        head = head -> next;
        temp1->next = NULL;
        temp->next = head;
        free(temp1);
    }
    printf("Deletion successful\n");

    return head;
}

void display(struct node *head){
    struct node *temp = head;
    int i = 1;
    do{
        printf("Node %d -> Data %d\n", i, temp->data);
        temp = temp -> next;
        i++;
    }while(temp != head);
}

int main(){
    int Case;
    struct node *head = NULL;

    do{
        printf("1. Create List\n");
        printf("2. Insert a node\n");
        printf("3. Delete a node\n");
        printf("4. Display list\n");
        printf("5. Exit\n");
        printf("Select choice: ");
        scanf("%d", &Case);

        switch (Case)
        {
        case 1:
            head = create(head);
            break;

        case 2:
            head = insert(head);
            break;

        case 3:
            head = delete(head);
            break;

        case 4:
            display(head);
            break;

        case 5:
            printf("Jaa rhe ho na chhod kar :(\nSab aesa hi karte hai :(\nGood Bye :(\n");
            break;

        default:
            printf("Watashinonamaeha kirayoshikage. 33-Sai\n");
            break;
        }
```

```
136.
137.     }while(Case != 5);
138.
139.     return 0;
140. }
141.
```

---

## DS PRACTICAL SLIPS/slip5.1.c

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

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

struct node *create(struct node *head){
    int n, data;
    struct node *temp = head;
    printf("How many node: ");
    scanf("%d", &n);

    for(int i = 0; i < n; i++){
        struct node *newnode = (struct node *)malloc(sizeof(struct node));
        printf("Enter data for node [%d]: ", i + 1);
        scanf("%d", &data);
        newnode -> data = data;
        newnode -> next = NULL;

        if(head == NULL){
            head = newnode;
            temp = newnode;
        }
        else{
            temp -> next = newnode;
            temp = newnode;
        }
    }
    return head;
}

void display(struct node *head){
    int i = 1;
    struct node *tr = head;
    while(tr != NULL){
        printf("Node %d -> Data: %d\n", i, tr->data);
        tr = tr -> next;
        i++;
    }
}

int main(){
    struct node *head = NULL;
    head = create(head);
    display(head);
    return 0;
}

```



---

## DS PRACTICAL SLIPS/slip5.2.c

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

```c

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

struct tree *create(int data){
    struct tree *newnode = (struct tree *)malloc(sizeof(struct tree));
    newnode->data = data;
    newnode->left = newnode->right = NULL;

    return newnode;
}

struct tree *insert(struct tree *root, int data){
    if(root == NULL){
        return create(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 inorder(struct tree *root){
    if(root != NULL){
        inorder(root->left);
        printf("%d ", root->data);
        inorder(root->right);
    }
}

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


int main(){
    struct tree *root = NULL;
    int c, n;
    printf("How many nodes: ");
    scanf("%d", &c);

    for(int i = 0; i < c; i++){
        printf("Enter element %d: ", i + 1);
        scanf("%d", &n);
        root = insert(root, n);
    }


    printf("Inorder: \n");
    inorder(root);
    printf("\n");

    printf("Preorder: \n");
    preorder(root);
    printf("\n");
```

```
70.
71.     return 0;
72.
73. }
74.
```

---

## DS PRACTICAL SLIPS/slip6.1.c

```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#define MAX 100

struct stack{
    int top;
    char arr[MAX];
};

void init(struct stack *s){
    s->top = -1;
}

int isFull(struct stack *s){
    return s->top == MAX - 1;
}

int isEmpty(struct stack *s){
    return s->top == -1;
}

void push(struct stack *s, char ch){
    if(!isFull(s)){
        s->arr[++(s->top)] = ch;
    }else{
        printf("Pani ki tanki bhar gayi hai, kripaya motor band karde\n");
        exit(1);
    }
}

char pop(struct stack *s){
    if(!isEmpty(s)){
        return s->arr[(s->top)--];
    }else{
        printf("Khali hai, khali hai\n");
        exit(1);
    }
}

void strrevs(char str[]){
    struct stack s;
    init(&s);

    int n = strlen(str);

    for(int i = 0; i < n; i++){
        push(&s, str[i]);
    }

    for(int i = 0; i < n; i++){
        str[i] = pop(&s);
    }
}

int main(){
    char str[MAX];
```

```
58.
59.     printf("Enter string: ");
60.     gets(str);
61.
62.     strrevs(str);
63.
64.     printf("Reversed string: \"%s\"\n", str);
65.
66.     return 0;
67. }
68.
```

---

**DS PRACTICAL SLIPS/slip6.2.c**

```
 1. #include<stdio.h>
 2. #include<stdlib.h>
 3.
 4. struct tree{
 5.     int data;
 6.     struct tree *left;
 7.     struct tree *right;
 8. };
 9.
10. struct tree *create(int data){
11.     struct tree *newnode = (struct tree *)malloc(sizeof(struct tree));
12.     newnode->data = data;
13.     newnode->left = newnode->right = NULL;
14.
15.     return newnode;
16. }
17.
18. struct tree *insert(struct tree *root, int data){
19.     if(root == NULL){
20.         return create(data);
21.     }
22.
23.     if(data < root->data){
24.         root->left = insert(root->left,data);
25.     }
26.     else if(data > root->data){
27.         root->right = insert(root->right, data);
28.     }
29.
30.     return root;
31. }
32.
33. int countNonLeaf(struct tree *root){
34.     if(root == NULL){
35.         printf("Tree is cutted down to logs\nBurned at fireplaces\nThe Sacrifies will be
remembered by the Generation\n");
36.         return 0;
37.     }
38.     if(root->left == NULL && root->right == NULL){
39.         return 0;
40.     }
41.     else{
42.         return (1 + countNonLeaf(root->left) + countNonLeaf(root->right));
43.     }
44. }
45.
46. int countLeaf(struct tree *root){
47.     if(root == NULL){
48.         return 0;
49.     }
50.     else if((root->left == NULL) && (root->right == NULL)){
```

```c
        return 1;
    }
    else{
        return(countLeaf(root->left) + countLeaf(root->right));
    }
}

int main(){
    struct tree *root = NULL;
    int ch;
    int c, n;
    int gg;
    int j;
    int k;
    printf("How many nodes: ");
    scanf("%d", &c);

    for(int i = 0; i < c; i++){
        printf("Enter element for node %d: ", i + 1);
        scanf("%d", &n);
        root = insert(root, n);
    }

    do{
        printf("1. Insert new element in tree\n");
        printf("2. Count total non-leaf nodes\n");
        printf("3. count total leaf nodes\n");
        printf("0. Exit\n");
        printf("Select choice: ");
        scanf("%d", &ch);

        switch (ch)
        {
        case 1:
            printf("Enter element: ");
            scanf("%d", &gg);
            root = insert(root, gg);
            break;

        case 2:
            j = countNonLeaf(root);
            printf("Total non-leaf node: %d\n", j);
            break;

        case 3:
            k = countLeaf(root);
            printf("Total leaf node: %d\n", k);
            break;

        case 0:
            printf("Kaisi teri khud garzi naa dhoop chune na chava\n");
            break;

        default:
            printf("Re kabiraaaaa.... maaan jaa.... thik se dekh k button daba.....\n");
            break;
        }

    }while(ch != 0);

    return 0;
}
```



---

**DS PRACTICAL SLIPS/slip7.1.c**

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

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

struct node *ins(struct node *head){
    struct node *newnode = (struct node *)malloc(sizeof(struct node));
    newnode->next = NULL;

    printf("Enter data: ");
    scanf("%d", &newnode->data);

    if(head == NULL){
        newnode->next = NULL;
        head = newnode;
    }
    else{
        newnode->next = head;
        head = newnode;
    }
    printf("Insertion successful\n");
    return head;
}

void disp(struct node *head){
    struct node *tr = head;
    int i = 1;
    while(tr!=NULL){
        printf("Node: %d -> Data: %d\n", i, tr->data);
        tr = tr -> next;
        i++;
    }
}

int lent(struct node *head){
    struct node *tr = head;
    int lens = 1;
    while(tr != NULL){
        lens++;
        tr = tr -> next;
    }

    return lens;
}

int main(){
    struct node *head = NULL;
    int ch;
    int len;
    do{
        printf("1. Create/Insert node\n");
        printf("2. Display list\n");
        printf("3. Length of list\n");
        printf("0. Muze kaise bhi karke nikal mere bhai me fass gaya hun yaha...\n");
        printf("Enter choice: ");
        scanf("%d", &ch);

        switch(ch){
            case 1:
                head = ins(head);
                break;

            case 2:
                disp(head);
```

```
                break;

            case 3:
                len = lent(head);
                printf("Length of list: [%d]\n", len);
                break;

            case 0:
                printf("Sayonara\n");
                break;

            default:
                printf("Bhaisahab ye kis line me aa gaye aap? \n");
                break;
        }

    }while(ch!=0);

    return 0;
}

```

---

**DS PRACTICAL SLIPS/slip7.2.c**

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

struct Stack {
    int *arr;
    int top;
    int capacity;
};

struct Stack* createStack(int capacity) {
    struct Stack *stack = (struct Stack *)malloc(sizeof(struct Stack));
    stack->capacity = capacity;
    stack->top = -1;
    stack->arr = (int *)malloc(stack->capacity * sizeof(int));
    return stack;
}

int isEmpty(struct Stack *stack) {
    return stack->top == -1;
}

int isFull(struct Stack *stack) {
    return stack->top == stack->capacity - 1;
}

void push(struct Stack *stack, int data) {
    if (isFull(stack)) {
        printf("Stack overflow! Cannot push %d\n", data);
        return;
    }
    stack->top++;
    stack->arr[stack->top] = data;
    printf("%d pushed to stack\n", data);
}

int pop(struct Stack *stack) {
    if (isEmpty(stack)) {
        printf("Stack underflow! Nothing to pop\n");
        return -1;
    }
    int popped = stack->arr[stack->top];
```

```c
    stack->top--;
    return popped;
}

void display(struct Stack *stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty\n");
        return;
    }
    printf("Stack elements (top -> bottom): ");
    for (int i = stack->top; i >= 0; i--) {
        printf("%d ", stack->arr[i]);
    }
    printf("\n");
}

int main() {
    int capacity;
    printf("Enter stack capacity: ");
    scanf("%d", &capacity);

    struct Stack *stack = createStack(capacity);

    int choice, value;

    do {
        printf("\n--- Stack Menu ---\n");
        printf("1. Push\n");
        printf("2. Pop\n");
        printf("3. Display\n");
        printf("4. IsEmpty\n");
        printf("5. IsFull\n");
        printf("0. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch(choice) {
            case 1:
                printf("Enter value to push: ");
                scanf("%d", &value);
                push(stack, value);
                break;
            case 2:
                value = pop(stack);
                if (value != -1)
                    printf("Popped element: %d\n", value);
                break;
            case 3:
                display(stack);
                break;
            case 4:
                if (isEmpty(stack))
                    printf("Stack is empty\n");
                else
                    printf("Stack is not empty\n");
                break;
            case 5:
                if (isFull(stack))
                    printf("Stack is full\n");
                else
                    printf("Stack is not full\n");
                break;
            case 0:
                printf("Jaa raha hun mein....\n");
                break;
            default:
                printf("Chalo galti insaan se hi hoti hai...\n");
```

```
        }

    } while(choice != 0);

    return 0;
}

```

---

```
DS PRACTICAL SLIPS/slip8.1.c
#include<stdio.h>
#include<stdlib.h>

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

struct node *create(struct node *head){
    int data;
    int val;
    struct node *temp = head;

    printf("How many nodes: ");
    scanf("%d", &val);

    for(int i = 0; i < val; i++){
        struct node *newnode = (struct node *)malloc(sizeof(struct node));
        printf("Enter data for node %d: ", i + 1);
        scanf("%d", &data);

        newnode->data = data;
        newnode->prev = newnode->next = NULL;

        if(head == NULL){
            head = newnode;
            temp = newnode;
        }
        else{
            temp->next = newnode;
            newnode->prev = temp;
            temp = newnode;
        }
    }
    printf("Creation successfull\n");

    return head;
}

struct node *display(struct node *head){
    struct node *tr = head;
    int i = 1;
    while(tr != NULL){
        printf("Node %d -> data %d\n", i, tr->data);
        tr = tr ->next;
        i++;
    }
}

int main(){
    struct node *head = NULL;

    head = create(head);
```

```
    display(head);

    return 0;
}

```

---

## DS PRACTICAL SLIPS/slip8.2.c

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

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

struct node *create(struct node *head){
    int data;
    int val;
    struct node *temp = head;
    printf("How many nodes: ");
    scanf("%d", &val);
    for(int i = 0; i < val; i++){
        struct node *newnode = (struct node *)malloc(sizeof(struct node));
        printf("Insert data for node [%d]: ", i + 1);
        scanf("%d", &data);

        newnode->data = data;
        newnode->next = NULL;

        if(head == NULL){
            head = newnode;
            temp = newnode;
        }
        else{
            temp->next = newnode;
            temp = newnode;
        }
    }
    return head;
}

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

    return head1;
}

void display(struct node *head){
    int i = 1;
    struct node *tr = head;
    while (tr != NULL)
    {
        printf("Node %d -> Data: [%d]\n", i, tr->data);
        tr = tr -> next;
        i++;
    }
}

int main(){
    struct node *head1 = NULL;
```

```c
    struct node *head2 = NULL;

    printf("Initializing process for list 1\n");
    head1 = create(head1);
    printf("Displaying list 1: \n");
    display(head1);

    printf("Initializing process for list 2\n");
    head2 = create(head2);
    printf("Displaying list 2: \n");
    display(head2);

    printf("Concatinating both lists....\n");
    printf("Taal se Taal milaaa\n");
    concat(head1, head2);
    printf("Concatination successful\n");
    printf("Displaying concatinated list\n");
    display(head1);

    return 0;

}

```

---

## DS PRACTICAL SLIPS/slip9.1.c

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

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

struct node *create(struct node *head){
    int data;
    int val;
    struct node *temp = head;
    printf("How many nodes: ");
    scanf("%d", &val);
    for(int i = 0; i < val; i++){
        struct node *newnode = (struct node *)malloc(sizeof(struct node));
        printf("Insert data for node [%d]: ", i + 1);
        scanf("%d", &data);

        newnode->data = data;
        newnode->next = NULL;

        if(head == NULL){
            head = newnode;
            temp = newnode;
        }
        else{
            temp->next = newnode;
            temp = newnode;
        }
    }
    return head;
}

void display(struct node *head){
    int i = 1;
    struct node *tr = head;
    while (tr != NULL)
    {
```

```c
        printf("Node %d -> Data: [%d]\n", i, tr->data);
        tr = tr -> next;
        i++;
    }
}

struct node *noderev(struct node *head){
    struct node *prev = NULL;
    struct node *curr = head;
    struct node *next = NULL;

    while(curr != NULL){
        next = curr -> next;
        curr->next = prev;
        prev = curr;
        curr = next;
    }

    return prev;
}

int main(){
    struct node *head = NULL;
    head = create(head);
    display(head);

    printf("In Reverse order: \n");
    head = noderev(head);
    display(head);

    return 0;
}

```

---

**DS PRACTICAL SLIPS/slip9.2.c**

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

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

struct node *create(struct node *head)
{
    int data;
    int val;
    struct node *temp = head;
    printf("How many nodes: ");
    scanf("%d", &val);
    for (int i = 0; i < val; i++)
    {
        struct node *newnode = (struct node *)malloc(sizeof(struct node));
        printf("Insert data for node [%d]: ", i + 1);
        scanf("%d", &data);

        newnode->data = data;
        newnode->next = NULL;

        if (head == NULL)
        {
            head = newnode;
            temp = newnode;
```

```
        }
        else
        {
            temp->next = newnode;
            temp = newnode;
        }
    }
    return head;
}

void display(struct node *head)
{
    int i = 1;
    struct node *tr = head;
    while (tr != NULL)
    {
        printf("Node %d -> Data: [%d]\n", i, tr->data);
        tr = tr->next;
        i++;
    }
}

void nodeser(struct node *head)
{
    struct node *temp = head;
    int n;
    int flag = 0;
    int pos = 1;
    printf("Enter element to search: ");
    scanf("%d", &n);

    while (temp != NULL)
    {
        if (temp->data == n)
        {
            flag = 1;
        }
        temp = temp->next;
        pos++;
    }
    if (flag == 1)
    {
        printf("%d is in %d node\n", n, pos);
    }
    else
    {
        printf("%d is not in list\n");
    }
}

struct node* deleteNode(struct node *head) {
    int pos;
    struct node *temp = head;
    printf("Enter node number to delete: ");
    scanf("%d", &pos);

    if (pos == 1) {
        head = head->next;
        free(temp);
        printf("Node %d deleted.\n", pos);
        return head;
    }

    struct node *prev = NULL;
    int i = 1;
    while (temp != NULL && i < pos) {
        prev = temp;
```

```
        temp = temp->next;
        i++;
    }

    prev->next = temp->next;
    free(temp);
    printf("Node %d deleted.\n", pos);

    return head;
}


int main()
{
    struct node *head = NULL;
    int sel;

    while (1)
    {
        printf("\nMenu:\n");
        printf("1. Create Linked List\n");
        printf("2. Display Linked List\n");
        printf("3. Search Node\n");
        printf("4. Delete from specific pos\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &sel);

        switch (sel)
        {
        case 1:
            head = create(head);
            break;
        case 2:
            display(head);
            break;
        case 3:
            nodeser(head);
            break;
        case 4:
            head = deleteNode(head);
            break;

        case 5:
            exit(0);
            break;

        default:
            printf("Galat baat!\n");
        }
    }

    return 0;
}

```



---

## DS PRACTICAL SLIPS/slip10.1.c

```
#include<stdio.h>

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

```

```c
    int graph[n][n];
    printf("Enter adjancency matrix: \n");
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            scanf("%d", &graph[i][j]);
        }
    }

    for(int i = 0; i < n; i++){
        int indeg = 0; int outdeg = 0;
        for(int j = 0; j < n; j++){
            outdeg = outdeg + graph[i][j];
            indeg = indeg + graph[j][i];
        }
        printf("Node %d -> Indegree: [%d] || Outdegree: [%d]\n", i, indeg, outdeg);
    }
    return 0;
}

DS PRACTICAL SLIPS/slip10.2.c
#include<stdio.h>
#include<stdlib.h>

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

struct node *queue[100];
int fr = -1, rear = -1;

struct node *create(int data){
    struct node *newnode = (struct node *)malloc(sizeof(struct node));
    newnode->data = data;
    newnode->left = newnode->right = NULL;

    return newnode;
}

struct node *insert(struct node *root, int data){
    if(root == NULL){
        return create(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 enqueue(struct node *nodule){
    if(nodule != NULL){
        if(rear == 99) return;
        queue[++rear] = nodule;
    }
}

struct node *dequeue(){
    if(fr == rear){
        return NULL;
    }
    return queue[++fr];
```

```c
}

void levelOrder(struct node *root){
    if(root == NULL){
        return;
    }
    enqueue(root);
    while(fr != rear){
        struct node *curr = dequeue();
        printf("[%d]\t", curr->data);
        enqueue(curr->left);
        enqueue(curr->right);
    }
}

int main(){
    int n, c;
    struct node *root = NULL;
    printf("How many nodes: ");
    scanf("%d", &c);
    for(int i = 0; i < c; i++){
        printf("Enter data for node %d: ", i + 1);
        scanf("%d", &n);
        root = insert(root, n);
    }
    printf("Displaying tree: \n");
    levelOrder(root);

    return 0;
}

```

---

**DS PRACTICAL SLIPS/slip11.1.c**

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

void disp(int *arr, int n){
    for(int i = 0; i < n; i++){
        printf("[%d]\t", arr[i]);
    }
    printf("\n");
}

void swap(int *a, int *b){
    int t = *a;
    *a = *b;
    *b = t;
}

int party(int *arr, int low, int high){
    int piv = arr[low];
    int i = low + 1;
    int j = high;

    while(1){
        while(i <= high && arr[i] <= piv){
            i++;
        }

        while (arr[j] > piv){
            j--;
        }

```

```c
        if(i < j){
            swap(&arr[i], &arr[j]);
        }else{
            break;
        }

    }

    swap(&arr[low], &arr[j]);
    return j;

}

void quick(int *arr, int low, int high){
    if(low < high){
        int pi = party(arr, low, high);
        quick(arr, low, pi - 1);
        quick(arr, pi + 1, high);
    }
}

int main(){
    int n;
    printf("Enter n: ");
    scanf("%d", &n);
    int arr[n];
    srand(time(NULL));
    printf("Entering %d elements: \n", n);
    for(int i = 0; i < n; i++){
        arr[i] = rand();
    }
    printf("Insertion successful\n");
    printf("Displaying: \n");
    disp(arr, n);
    quick(arr, 0, n - 1);
    printf("After sort: \n");
    disp(arr, n);

    return 0;
}

```

---

## DS PRACTICAL SLIPS/slip11.2.c

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

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

struct tree *create(int data){
    struct tree *newnode = (struct tree *)malloc(sizeof(struct tree));

    newnode->data = data;
    newnode->left = newnode->right = NULL;

    return newnode;
}

struct tree *insert(struct tree *root, int data){
```

```c
    if(root == NULL){
        return create(data);
    }

    if(data < root -> data){
        root -> left = insert(root->left, data);
    }

    else if(data > root -> data){
        root -> right = insert(root->right, data);
    }

    return root;
}

int leafNode(struct tree *root){
    if(root == NULL){
        return 0;
    }
    else if(root->left == NULL && root->right == NULL){
        return 1;
    }
    else{
        return (leafNode(root->left) + leafNode(root->right));
    }
}

int nonLeafNode(struct tree *root){
    if(root == NULL){
        return 0;
    }
    else if(root->left == NULL && root->right == NULL){
        return 0 ;
    }
    else{
        return (1 + nonLeafNode(root->left) + nonLeafNode(root->right));
    }
}

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

}

int main(){
    int n, c;
    struct tree *root = NULL;
    printf("Enter n: ");
    scanf("%d", &c);

    for(int i = 0; i < c; i++){
        printf("Enter data for node %d: ", i + 1);
        scanf("%d", &n);
        root = insert(root, n);
    }

    printf("Displaying tree: \n");
    inorder(root);

    int i = leafNode(root);
    int j = nonLeafNode(root);

    printf("\nTotal leaf node: [%d]\n", i);
```

```
    printf("\nTotal non leaf node: [%d]\n", j);

    return 0;
}

DS PRACTICAL SLIPS/slip12.1.c
#include <stdio.h>

int main() {
    int deg1, deg2, maxDeg;

    // Input first polynomial
    printf("Enter degree of first polynomial: ");
    scanf("%d", °1);
    int poly1[deg1 + 1];
    printf("Enter coefficients of first polynomial (from highest degree to constant):\n");
    for (int i = deg1; i >= 0; i--) {
        printf("Coefficient of x^%d: ", i);
        scanf("%d", &poly1[i]);
    }

    // Input second polynomial
    printf("\nEnter degree of second polynomial: ");
    scanf("%d", °2);
    int poly2[deg2 + 1];
    printf("Enter coefficients of second polynomial (from highest degree to constant):\n");
    for (int i = deg2; i >= 0; i--) {
        printf("Coefficient of x^%d: ", i);
        scanf("%d", &poly2[i]);
    }

    // Determine maximum degree
    maxDeg = (deg1 > deg2) ? deg1 : deg2;

    // Arrays to hold result
    int result[maxDeg + 1];

    // Initialize result
    for (int i = 0; i <= maxDeg; i++) {
        int a = (i <= deg1) ? poly1[i] : 0;
        int b = (i <= deg2) ? poly2[i] : 0;
        result[i] = a - b;
    }

    // Display result polynomial
    printf("\nResult (P1 - P2) = ");
    for (int i = maxDeg; i >= 0; i--) {
        if (result[i] != 0) {
            if (i != maxDeg && result[i] > 0) printf("+"); // add + for positive terms
            if (i == 0) printf("%d ", result[i]);          // constant
            else if (i == 1) printf("%dx ", result[i]);    // x term
            else printf("%dx^%d ", result[i], i);         // x^i term
        }
    }
    printf("\n");

    return 0;
}
```

---

## DS PRACTICAL SLIPS/slip12.2.c

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

```

```c
int queue[SiZe];
int fr = -1, rr = -1;

void insert(int value){
    if(fr == -1){
        fr = rr = 0;
    }else{
        rr = (rr + 1) % SiZe;
    }

    queue[rr] = value;
    printf("INSERTION SUCESSFULL\n");
}

void delete(){
    int deleted = queue[fr];
    if(fr == rr){
        fr = rr = -1;
    }else{
        fr = (fr + 1) % SiZe;
    }
    printf("DELETION SUCCESSFULL\n");
}

void disp(){
    int i = fr;
    while(1){
        printf("%d ", queue[i]);
        if(i == rr)
            break;

        i = (i + 1) % SiZe;
    }
    printf("\n");
}

int main(){
    int val, ch;

    while(1){
        printf("1. Insert element\n");
        printf("2. Delete element\n");
        printf("3. Display Queue\n");
        printf("4. Exit\n");
        printf("Your choice: ");
        scanf("%d", &ch);

        switch(ch){
            case 1:
                printf("Enter value to insert: ");
                scanf("%d", &val);
                insert(val);
                break;

            case 2:
                delete();
                break;

            case 3:
                disp();
                break;

            case 4:
                return 0;

            default:
                printf("Galat jawab, ye aapke bank mese 7 crores mere bank mein\n");
```

```
                break;
            }
        }

        return 0;
}

```

---

## DS PRACTICAL SLIPS/slip13.1.c

```c
#include<stdio.h>

void disp(int *arr, int n){
    for(int i = 0; i < n; i++){
        printf("[%d]\t", arr[i]);
    }
    printf("\n");
}

void mergerSort(int *arr, int low, int mid, int high){
    int copArr[100];
    int i = low;
    int j = mid + 1;
    int k = low;

    while(i <= mid && j <= high){
        if(arr[i] < arr[j]){
            copArr[k] = arr[i];
            i++;
        }
        else{
            copArr[k] = arr[j];
            j++;
        }
        k++;
    }
    while(i <= mid){
        copArr[k] = arr[i];
        k++;
        i++;
    }
    while(j <= high){
        copArr[k] = arr[j];
        k++;
        j++;
    }
    for(int l = 0; l <= high; l++){
        arr[l] = copArr[l];
    }
}

void merger(int *arr, int low, int high){
    int mid;
    if(low < high){
        mid = (low + high) / 2;
        merger(arr, low, mid);
        merger(arr, mid + 1, high);
        mergerSort(arr, low, mid, high);
    }
}

int main(){
    int n;
    printf("Enter n: ");
    scanf("%d", &n);
```

```
    int arr[n];
    printf("Enter %d elements: ", n);

    for(int i = 0; i < n; i++){
        scanf("%d", &arr[i]);
    }

    disp(arr, n);
    printf("After Sorting: \n");
    merger(arr, 0, n - 1);
    disp(arr, n);

    return 0;
}

```

---

## DS PRACTICAL SLIPS/slip13.2.c

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

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

struct node *create(struct node *head){
    int data;
    int n;
    struct node *temp = head;
    printf("How many nodes: ");
    scanf("%d", &n);
    for(int i = 0; i < n; i++){
        struct node *newnode = (struct node *)malloc(sizeof(struct node));
        printf("Enter data for node %d: ", i + 1);
        scanf("%d", &data);

        newnode->data = data;
        newnode->next = NULL;

        if(head == NULL){
            head = newnode;
            temp = newnode;
        }
        else{
            temp->next = newnode;
            temp = newnode;
        }
    }
    printf("Creation Successful\n");
    return head;
}

void display(struct node *head){
    int i = 1;
    struct node *tr = head;
    while(tr != NULL){
        printf("Node -> %d : Data -> %d\n", i, tr->data);
        tr = tr ->next;
        i++;
    }
}

int len(struct node *head){
    int i = 0;
```

```c
    struct node *tr = head;
    while(tr != NULL){
        tr = tr ->next;
        i++;
    }
    return i;
}

void sum(struct node *head){
    struct node *temp = head;
    int sum = 0;
    while(temp != NULL){
        sum = sum + temp->data;
        temp = temp -> next;
    }

    printf("The Sum of total elements in list is: [%d]\n", sum);
}

int main(){
    int n;
    struct node *head = NULL;

    while(1){
        printf("1. Create List\n");
        printf("2. Length of list\n");
        printf("3. Sum of data in list\n");
        printf("4. Display\n");
        printf("5. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &n);

        switch (n)
        {
        case 1:
            head = create(head);
            break;

        case 2:
            printf("The total length: %d\n", len(head));
            break;

        case 3:
            sum(head);
            break;

        case 4:
            display(head);
            break;

        case 5:
            return 0;

        default:
            printf("Wrong choice\nJust like ur ex\n");
            break;
        }

    }

    return 0;
}

```



---

**DS PRACTICAL SLIPS/slip14.1.c**

```c
#include<stdio.h>

int binSearch(int *arr, int low, int high, int key){

    if (low > high){
        return -1;
    }

    int mid = (low + high) / 2;

    if(arr[mid] == key){
        return mid;
    }
    else if(arr[mid] < key){
        return binSearch(arr, mid + 1, high, key);
    }
    else{
        return binSearch(arr, low, mid - 1, key);
    }
}

int main(){
    int n;
    int key;
    printf("Enter n: ");
    scanf("%d", &n);
    int arr[n];
    printf("Enter elements in sorted order: ");
    for(int i = 0; i < n; i++)
        scanf("%d", &arr[i]);

    printf("Enter element to search: ");
    scanf("%d", &key);

    int res = binSearch(arr, 0, n - 1, key);

    if(res != -1){
        printf("[%d] found at [%d] location\n", key, res);
    }
    else{
        printf("Not found / array is not in sorted order");
    }

    return 0;

}

```

---

**DS PRACTICAL SLIPS/slip14.2.c**

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

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

struct tree *create(int data){
    struct tree *newtree = (struct tree *)malloc(sizeof(struct tree));

    newtree->data = data;
    newtree->left = newtree->right = NULL;

```

```c
    return newtree;
}

struct tree *insert(struct tree *root, int data){
    if(root == NULL){
        return create(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 inorder(struct tree *root){
    if(root != NULL){
        inorder(root->left);
        printf("%d ", root->data);
        inorder(root->right);
    }
}

int countNode(struct tree *root){
    static int count;
    if(root == NULL){
        return 0;
    }
    else{
        count = 1 + countNode(root->left) + countNode(root->right);
    }

    return count;
}

int countOdd(struct tree *root){
    if(root == NULL){
        return 0;
    }


    int isOdd = ((root->data % 2 != 0) ? 1 : 0);

    return isOdd + countOdd(root->left) + countOdd(root->right);
}

int main(){
    int ch;
    int n, c;
    struct tree *root = NULL;

    printf("How many nodes: ");
    scanf("%d", &n);
    for(int i = 0; i < n; i++){
        printf("Enter data for node %d: ", i + 1);
        scanf("%d", &c);
        root = insert(root, c);
    }
    printf("\n");

    printf("Displaying: \n");
    inorder(root);
    printf("\n");
```

```c
    printf("Total no. of nodes: %d\n", countNode(root));
    printf("Total odd numbers: %d\n", countOdd(root));

    return 0;
}

```

---

## DS PRACTICAL SLIPS/slip15.1.c

```c
#include<stdio.h>

void disp(int *arr, int n){
    for(int i = 0; i < n; i++){
        printf("[%d]\t", arr[i]);
    }
    printf("\n");
}

void swap(int *a, int *b){
    int t = *a;
    *a = *b;
    *b = t;
}

int part(int *arr, int low, int high){
    int piv = arr[low];
    int i = low + 1;
    int j = high;

    while (1)
    {
        while(i <= high && arr[i] <= piv){
            i++;
        }

        while (arr[j] >= piv)
        {
            j--;
        }

        if(i<j){
            swap(&arr[i], &arr[j]);
        }else{
            break;
        }
    }

    swap(&arr[low], &arr[j]);
    return j;

}

void qckSort(int *arr, int low, int high){
    int pi;
    if(low < high){
        pi = part(arr, low, high);
        qckSort(arr, low, pi - 1);
        qckSort(arr, pi + 1, high);
    }
}

int main(){
    int n;
    printf("Enter n: ");
    scanf("%d", &n);
```

```c
    int arr[n];
    printf("Enter %d elements: ", n);
    for(int i = 0; i < n; i++){
        scanf("%d", &arr[i]);
    }

    disp(arr, n);
    printf("After Sort\n");
    qckSort(arr, 0, n - 1);
    disp(arr, n);

    return 0;
}

```

---

**DS PRACTICAL SLIPS/slip15.2.c**

```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#define MAX 100

struct stack{
    int top;
    int arr[MAX];
};

void init(struct stack *s){
    s->top = -1;
}

int isFull(struct stack *s){
    return (s->top == MAX - 1);
}

int isEmpty(struct stack *s){
    return (s->top == -1);
}

void push(struct stack *s, char ch){
    if(!isFull(s)){
        s->arr[++(s->top)] = ch;
    }
    else{
        printf("Stack Overflow\n");
        exit(1);
    }
}

char pop(struct stack *s){
    if(!isEmpty(s)){
        return s->arr[(s->top)--];
    }
    else{
        printf("Stack underflow \n");
        exit(1);
    }
}

void stringIsReversedByStackWOW(char str[]){
    struct stack s;
    init(&s);

```

```c
    int n = strlen(str);

    for(int i = 0; i < n; i++){
        push(&s, str[i]);
    }

    for(int i = 0; i < n; i++){
        str[i] = pop(&s);
    }
}

int main(){
    char str[MAX];
    printf("Enter string: ");
    gets(str);

    stringIsReversedByStackWOW(str);


    printf("Reversed String: %s\n", str);

    return 0;
}

```

---

## DS PRACTICAL SLIPS/slip16.1.c

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

void swap(char *a, char *b){
    char t = *a;
    *a = *b;
    *b = t;
}

void sorter(char str[], int n){
    int i, j;
    for(i = 0; i < n - 1; i++){
        for(j = 0; j < n - i - 1; j++){
            if(str[j] > str[j + 1]){
                swap(&str[j], &str[j + 1]);
            }
        }
    }
}

void disp(char str[], int n){
    int i;
    for(i = 0; i < n; i++){
        printf("[%c]", str[i]);
    }
    printf("\n");
}

int main(){
    char str[100];
    int n;

    printf("Enter string: ");
    gets(str);

```

```
    n = strlen(str);
    sorter(str, n);

    printf("After Sort:\n");
    disp(str, n);

    return 0;
}

```

---

## DS PRACTICAL SLIPS/slip16.2.c

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

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

struct queue *fr = NULL;
struct queue *rr = NULL;

struct queue *create(int val){
    struct queue *newnode = (struct queue *)malloc(sizeof(struct queue));
    newnode->data = val;
    newnode->next = NULL;

    return newnode;
}

void insert(int val){
    struct queue *newnode = create(val);
    if(fr == NULL){
        fr = rr = newnode;
    }
    else{
        rr->next = newnode;
        rr = newnode;
    }

    rr->next = fr;
    printf("INSERTION SUCCESSFUL\n");
}

void delete(){
    struct queue *temp;
    if(fr == NULL){
        printf("QUEUE EMPTY\n");
        return;
    }

    temp = fr;
    if(fr == rr){
        fr = NULL;
        rr = NULL;
    }else{
        fr = fr->next;
        rr->next = fr;
    }
    free(temp);
}

void display(){
    struct queue *temp;
```

```c

    if(fr == NULL){
        printf("Jane kaha mera jiya kidhar gaya ji...? Abhi abhi yahi tha kidhar gaya ji...?");
        return;
    }
    temp = fr;
    do{
        printf("[%d]->", temp->data);
        temp = temp->next;
    }while(temp != fr);

    printf("\n");
}

int main(){
    int ch, val;

    while(1){
        printf("1. Insert element\n");
        printf("2. Delete element\n");
        printf("3. Display\n");
        printf("4. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &ch);

        switch (ch)
        {
        case 1:
            printf("Enter number: ");
            scanf("%d", &val);
            insert(val);
            break;

        case 2:
            delete();
            break;

        case 3:
            display();
            break;

        case 4:
            return 0;

        default:
            printf("WRONG CHOICE\n");
            break;
        }
    }

    return 0;
}

```



---

**DS PRACTICAL SLIPS/slip17.1.c**

```c
#include<stdio.h>

struct cust{
    int cust_id;
    char name[50];
    char address[100];
};

```

```c
void disp(struct cust arr[], int n){
    for(int i = 0; i < n; i++){
        printf("\n");
        printf("Cust: %d\t\n", i + 1);
        printf("----------------------------------------\n");
        printf("ID:\t\t%d\n", arr[i].cust_id);
        printf("Name:\t\t%s\n", arr[i].name);
        printf("Address:\t%s\n", arr[i].address);
    }
    printf("----------------------------------------\n");
    printf("\n\n");
}

void sorterOP(struct cust arr[], int n){
    for(int i = 0; i < n - 1; i++){
        for(int j = 0; j < n - i - 1; j++){
            if(arr[j].cust_id > arr[j + 1].cust_id){
                struct cust temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}

int main(){
    int n;
    printf("How many customer: ");
    scanf("%d", &n);
    struct cust custumber[n];

    for(int i = 0; i < n; i++){
        printf("Enter details for cust %d\n\n", i + 1);
        printf("Enter customer id: ");
        scanf("%d", &custumber[i].cust_id);

        printf("Enter Name: ");
        scanf("%s", &custumber[i].name);

        printf("Enter address: ");
        scanf("%s", &custumber[i].address);
    }

    printf("Displaying list: \n");
    disp(custumber, n);
    printf("AFTER SORTING\n");
    sorterOP(custumber, n);
    disp(custumber, n);

    return 0;
}
```

---

## DS PRACTICAL SLIPS/slip17.2.c

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

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

struct node *create(int data){
```

```c
    struct node *newnode = (struct node *)malloc(sizeof(struct node));

    newnode->data = data;
    newnode->left = newnode->right = NULL;

    return newnode;
}

struct node *insert(struct node *root, int data){
    if(root == NULL){
        return create(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 inorder(struct node *root){
    if(root != NULL){
        inorder(root->left);
        printf("[%d]\t", root->data);
        inorder(root->right);
    }
}

int calcHight(struct node *root){
    if(root == NULL){
        return -1;
    }
    else{
        int lheight = calcHight(root->left);
        int rheight = calcHight(root->right);

        return(lheight > rheight ? lheight : rheight) + 1;
    }
}

int main(){
    int n, c;
    struct node *root = NULL;

    printf("How many nodes: ");
    scanf("%d", &n);

    for(int i = 0; i < n; i++){
        printf("Enter data for node %d: ", i + 1);
        scanf("%d", &c);

        root = insert(root, c);
    }

    printf("Displaying tree: \n");
    inorder(root);

    printf("\nHeight of tree: %d\n", calcHight(root));

    return 0;
}

```

## DS PRACTICAL SLIPS/slip18.1.c

```c
#include<stdio.h>
#include<ctype.h>
#include<string.h>

struct city{
    int city_code;
    char name[30];
};

int main(){
    int n;
    char str[30];
    int flag = 0;
    printf("Enter n: ");
    scanf("%d", &n);
    struct city c[n];

    printf("Enter city code and name\n");
    for(int i = 0; i < n; i++){
        printf("Enter data for city %d: \n", i + 1);
        printf("\n----------------------------------------\n");
        printf("Enter city code: ");
        scanf("%d", &c[i].city_code);

        printf("Enter city name: ");
        scanf("%s", &c[i].name);
        printf("\n----------------------------------------\n");
        printf("\n\n");
    }

    printf("Enter city to search: ");
    scanf("%s", str);

    for(int i = 0; i < n; i++){
        if(strcmp(c[i].name, str) == 0){
            printf("Code: [%d] : City [%s]", c[i].city_code, c[i].name);
            flag = 1;
        }
    }
    if(flag == 0){
        printf("%s is not present\n", str);
    }

    return 0;
}

```

---

## DS PRACTICAL SLIPS/slip18.2.c

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

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

struct queue *fr = NULL;
struct queue *rr = NULL;

struct queue *create(int val){
    struct queue *newnode = (struct queue *)malloc(sizeof(struct queue));
    newnode->data = val;
```

```c
    newnode->next = NULL;

    return newnode;
}

void insert(int val){
    struct queue *newnode = create(val);
    if(fr == NULL){
        fr = rr = newnode;
    }
    else{
        rr->next = newnode;
        rr = newnode;
    }

    printf("INSERTION SUCCSSFUL\n");
}

void delete(){
    struct queue *temp;
    if(fr == NULL){
        printf("Queue is empty\n");
        return;
    }
    temp = fr;
    if(fr == rr){
        fr = rr = NULL;
    }
    else{
        fr = fr -> next;
    }
    free(temp);
    printf("DELETION SUCCESSFUL\n");
}

void display(){
    struct queue *temp;

    if(fr == NULL){
        printf("Accha ji me hara chalo maan jaao na...\n");
        return;
    }
    temp = fr;

    while(temp != NULL){
        printf("[%d]->", temp->data);
        temp = temp->next;
    }
    printf("\n\a");
}

void length(){
    struct queue *temp = fr;
    int i = 1;
    while(temp != NULL){
        i++;
        temp = temp -> next;
    }
    printf("Length of queue: [%d]\n", i - 1);
}

void search(){
    struct queue *temp = fr;
    int n;
    int pos = 1;
    int flag = 0;
    printf("Enter element to search: ");
```

```c
    scanf("%d", &n);

    while (temp != NULL)
    {
        if(n == temp->data){
            flag = 1;
            pos++;
        }
        temp = temp -> next;
    }

    if(flag == 1){
        printf("%d found at %d location\n", n, pos + 1);
    }
    else{
        printf("%d is not in queue\n");
    }
}

int main(){
    int val;
    int n;

    while(1){
        printf("1. Insert element\n");
        printf("2. Delete element\n");
        printf("3. Display queue\n");
        printf("4. Length of queue\n");
        printf("5. Search elemnent in queue\n");
        printf("6. Exit\n");
        printf("Your choice: ");
        scanf("%d", &n);
        switch (n)
        {
        case 1:
            printf("Enter value: ");
            scanf("%d", &val);
            insert(val);
            break;

        case 2:
            delete();
            break;

        case 3:
            display();
            break;

        case 4:
            length();
            break;

        case 5:
            search();
            break;

        case 6:
            return 0;

        default:
            printf("galat kadam\nEk kadam swatchata ki ore\nswatch bharat abhiyaan me apna
sahyong de\n");
            break;
        }
    }
    return 0;
}
```

```
148.
```

---

## DS PRACTICAL SLIPS/slip19.1.c

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

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

void insertAtEnd(struct Node** head, char newData) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed!\n");
        return;
    }
    newNode->data = newData;
    newNode->next = NULL;

    if (*head == NULL) {
        *head = newNode;
        return;
    }

    struct Node* current = *head;
    while (current->next != NULL) {
        current = current->next;
    }

    current->next = newNode;
}

void displayList(struct Node* head) {
    printf("Vowels Linked List: ");
    struct Node* current = head;

    while (current != NULL) {
        printf("'%c' -> ", current->data);
        current = current->next;
    }
    printf("NULL\n");
}

void freeList(struct Node* head) {
    struct Node* current = head;
    struct Node* nextNode;

    while (current != NULL) {
        nextNode = current->next;
        free(current);
        current = nextNode;
    }
}

int main() {
    struct Node* head = NULL;

    printf("--- Creating the Linked List of Vowels ---\n");

    insertAtEnd(&head, 'a');
    insertAtEnd(&head, 'e');
    insertAtEnd(&head, 'i');
    insertAtEnd(&head, 'o');
```

```c
    insertAtEnd(&head, 'u');

    printf("\n--- Displaying the Linked List ---\n");
    displayList(head);

    freeList(head);

    return 0;
}

```

---

## DS PRACTICAL SLIPS/slip19.2.c

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

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

struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));

    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

void push(struct Node** top, int data) {
    struct Node* newNode = createNode(data);
    newNode->next = *top;
    *top = newNode;
    printf("Pushed %d onto the stack.\n", data);
}

int pop(struct Node** top) {
    if (*top == NULL) {
        printf("Stack Underflow: Stack is empty.\n");
        return -1;
    }
    struct Node* temp = *top;
    int data = temp->data;
    *top = (*top)->next;
    free(temp);
    return data;
}

void displayStack(struct Node* top) {
    if (top == NULL) {
        printf("Stack is empty.\n");
        return;
    }
    printf("Stack (TOP -> BOTTOM): ");
    struct Node* current = top;
    while (current != NULL) {
        printf("%d ", current->data);
        current = current->next;
    }
    printf("\n");
}

void reverseDisplay(struct Node* top) {
    if (top == NULL) {
        return;
```

```c
    }
    reverseDisplay(top->next);
    printf("%d ", top->data);
}

int main() {
    struct Node* top = NULL;
    int choice, data;

    while (1) {
        printf("\n--- Menu ---\n");
        printf("1. Push\n");
        printf("2. Pop\n");
        printf("3. Display (Top to Bottom)\n");
        printf("4. Reverse Display (Bottom to Top)\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");

        if (scanf("%d", &choice) != 1) {
            while (getchar() != '\n');
            printf("Invalid input. Please enter a number.\n");
            continue;
        }

        switch (choice) {
            case 1:
                printf("Enter data to push: ");
                scanf("%d", &data);
                push(&top, data);
                break;

            case 2:
                data = pop(&top);
                if (data != -1) {
                    printf("Popped element: %d\n", data);
                }
                break;

            case 3:
                displayStack(top);
                break;

            case 4:
                printf("Stack in Reverse Order (Insertion Order): ");
                reverseDisplay(top);
                printf("\n");
                break;

            case 5:
                printf("Exiting program.\n");
                freeStack(top);
                return 0;

            default:
                printf("Invalid choice. Please try again.\n");
        }
    }
}

```

## DS PRACTICAL SLIPS/slip20.1.c

```c
#include<stdio.h>

void display(int *arr, int n){
    for(int i = 0; i < n; i++){
        printf("[%d]\t", arr[i]);
    }
    printf("\n");
}

void insertionSort(int *arr, int n){
    int i, j, key;
    for(int i = 1; i < n;  i++){
        key = arr[i];
        j = i - 1;
        while(j >= 0 && arr[j] >= key){
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;
    }
}

int main(){
    int n;
    printf("Enter n: ");
    scanf("%d", &n);
    int arr[n];
    printf("Enter %d elements: ", n);
    for (int i = 0; i < n; i++)
    {
        scanf("%d", &arr[i]);
    }

    printf("\n--BEFORE SORT--\n");
    display(arr, n);
    insertionSort(arr, n);
    printf("\n--AFTER SORT--\n");
    display(arr, n);

    return 0;
}

```

---

## DS PRACTICAL SLIPS/slip20.2.c

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

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

struct node *create(int data){
    struct node *newnode = (struct node *)malloc(sizeof(struct node));
    newnode->data = data;
    newnode->left = newnode->right = NULL;

    return newnode;
}

struct node *insert(struct node *root, int data){
```

```c
    if(root == NULL){
        return create(data);
    }

    if(data > root -> data){
        root->right = insert(root->right, data);
    }
    else if(data < root -> data){
        root->left = insert(root->left, data);
    }

    return root;
}

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

int main(){
    int n, c;
    struct node *root = NULL;
    printf("How many nodes: ");
    scanf("%d", &c);

    for(int i = 0; i < c; i++){
        printf("Enter data for node %d: ", i + 1);
        scanf("%d", &n);
        root = insert(root, n);
    }

    inorder(root);

    return 0;
}
```



<END>