

DSA PRACTICAL FILE

LAKSSHAY SEHRAWAT
2021UCA1802
COCS002
CSAI - 1

Index -

- 1) Program to find the mean and the median of the numbers stored in an array.
- 2) Program to insert one element in an array and delete an element from an array
- 3) Program to search for a number in an array.
- 4) Program to sort an array.
- 5) Program to merge two sorted arrays
- 6) Program to store the marks obtained by 10 students in 5 courses in a two-dimensional array.
- 7) Program to implement a linked list.
- 8) Program to insert a node in a linked list and delete a node from a linked list.
- 9) Program to print the elements of a linked list in reverse order without disturbing the linked list.
- 10) Program to reverse a linked list.
- 11) Program to add two polynomials using linked list.
- 12) Program to implement a doubly linked list
- 13) Program to implement a stack using an array.
- 14) Program to implement a stack using a linked list.
- 15) Program to implement a queue using an array.
- 16) Program to implement a queue using a linked list
- 17) Program to implement a circular queue using a linked list.
- 18) Program to implement a priority queue using a linked list
- 19) Program to implement a double-ended queue using a linked list.
- 20) Program to construct a binary tree and display its preorder, inorder and postorder traversals
- 21) Program to construct a binary search tree
- 22) Program to construct a graph
- 23) Program to calculate the distance between two vertices in a graph
- 24) Program to calculate the distance between two vertices in a graph
- 25) Program to construct a minimal spanning tree of a graph.

Q1. Write a program to find the mean and the median of the numbers stored in an array.

Code:

```
#include <stdio.h>

int main() {

    float marks[] = {20,90,12,59,30,44,99,19,39,73,85};
    int len = sizeof(marks)/sizeof(float);

    // mean
    int total = 0;
    for (int i = 0; i < len; ++i) {
        total += marks[i];
    }
    printf("The means marks are: %.2f\n", total / (float) len);

    // median
    // 1. sorting
    float temp;
    for (int i = 0; i < len - 1; i++) {
        for (int j = 0; j < len - i - 1; j++) {
            if (marks[j] > marks[j+1]) {
                temp = marks[j];
                marks[j] = marks[j+1];
                marks[j+1] = temp;
            }
        }
    }
    // 2. finding median
    if (len % 2 == 0) {
        temp = (marks[len/2 - 1] + marks[len/2])/2;
    } else {
        temp = marks[(len - 1)/2];
    }
    printf("The median marks are: %.2f", temp);
    return 0;
}
```

Output:

```
The means marks are: 51.82
The median marks are: 44.00
[Program finished]
```

Q2. Write a program to insert one element in an array and delete an element from an array.

Code:

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 10
```

```
int main() {

    int *ar;
    ar = (int *) malloc(SIZE * 4);
    for (int i = 0; i < SIZE; i++) {
        ar[i] = i+1;
    }
```

```

puts("\nThe array before editing:");
for (int i = 0; i < SIZE; i++) {
    printf("%d ", ar[i]);
} puts("\n");

//deletion
int ind;
printf("Enter index to the number to be deleted: ");
scanf("%d", &ind);

if (ind < SIZE) {
    for (int i = ind; i < SIZE - 1; i++) {
        ar[i] = ar[i+1];
    }
}

ar = (int *) realloc(ar, 4 * (SIZE - 1));

puts("\nAfter deletion:");
for (int i = 0; i < SIZE - 1; i++) {
    printf("%d ", ar[i]);
} puts("\n");

int val;
printf("Enter value to be inserted: ");
scanf("%d", &val);
printf("Enter index to be inserted at: ");
scanf("%d", &ind);

ar = (int *) realloc(ar, SIZE * 4);
if (ind < SIZE) {
    for (int i = SIZE - 1; i > ind; i--) {
        ar[i] = ar[i-1];
    }
} ar[ind] = val;

puts("\nAfter insertion:");
for (int i = 0; i < SIZE; i++) {
    printf("%d ", ar[i]);
}
}

```

Output:

```

The array before editing:
1 2 3 4 5 6 7 8 9 10

Enter index to the number to be deleted: 5

After deletion:
1 2 3 4 5 7 8 9 10

Enter value to be inserted: 6
Enter index to be inserted at: 5

After insertion:
1 2 3 4 5 6 7 8 9 10
[Program finished]

```

Q3. Write a program to search for a number in an array.

Code:

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

```

int main() {
    int arr[] = {0,23,24,9,55,334,26,29,90,73,320}, val;
    int len = sizeof(arr)/sizeof(int);
    int x = len;
    printf("Enter value to be searched: ");
    scanf("%d", &val);
    printf("%d", val);
    for (int i = 0; i < len; i++){
        if (arr[i] == val) {
            x = i;
            break;
        }
    }
    x == len ? printf(" is not in the array.") :
    printf(" is at index number %d", x);
}

```

Output:

```

Enter value to be searched: 23
23 is at index number 1
[Program finished]

```

Q4. Write a program to sort an array.

Code:

```

#include <stdio.h>

void main()
{
    int i, j, a, n;
    printf("Enter the value of N \n");
    scanf("%d", &n);
    int number[n];
    printf("Enter the numbers \n");
    for (i = 0; i < n; ++i)
        scanf("%d", &number[i]);

```

```

for (i = 0; i < n; ++i)
{
    for (j = i + 1; j < n; ++j)
    {
        if (number[i] > number[j])
        {
            a = number[i];
            number[i] = number[j];
            number[j] = a;
        }
    }
}

printf("The numbers arranged in ascending order are given below \n");
for (i = 0; i < n; ++i)
    printf("%d\n", number[i]);
}

```

Output:

```

Enter the value of N
5
Enter the numbers
5
4
3
2
1
The numbers arranged in ascending order are given below
1
2
3
4
5
[Program finished]

```

Q5. Write a program to merge two sorted arrays.

Code:

```

#include <stdio.h>

void main()
{

```

```
int m, n, i, j, k = 0;
printf("\n Enter size of array Array 1: ");
scanf("%d", &m);

int array1[m];

printf("\n Enter sorted elements of array 1: \n");
for (i = 0; i < m; i++)
{
    scanf("%d", &array1[i]);
}

printf("\n Enter size of array 2: ");
scanf("%d", &n);

int array2[n], array3[m+n];

printf("\n Enter sorted elements of array 2: \n");
for (i = 0; i < n; i++)
{
    scanf("%d", &array2[i]);
}

i = 0;
j = 0;

while (i < m && j < n)
{
    if (array1[i] < array2[j])
    {
        array3[k] = array1[i];
        i++;
    }
}
```

```
        else
        {
            array3[k] = array2[j];
            j++;
        }
        k++;
    }

    if (i >= m)
    {
        while (j < n)
        {
            array3[k] = array2[j];
            j++;
            k++;
        }
    }

    if (j >= n)
    {
        while (i < m)
        {
            array3[k] = array1[i];
            i++;
            k++;
        }
    }

    printf("\n After merging: \n");
    for (i = 0; i < m + n; i++)
    {
        printf("\n%d", array3[i]);
    }

}
```

Output:

```
Enter size of array Array 1: 3
Enter sorted elements of array 1:
1
2
3

Enter size of array 2: 2
Enter sorted elements of array 2:
4
5

After merging:
1
2
3
4
5
[Program finished]
```

Q6. Write a program to store the marks obtained by 10 students in 5 courses in a two-dimensional array.

Code:

```
#include <stdio.h>

int main() {
    float score[5][10];
    int max;
    printf("Enter the max marks: ");
    scanf("%d", &max);

    for (int i = 0; i < 5; i++) {
        printf("\nEnter the marks for subject %d\n", i+1);
        for (int j = 0; j < 10; j++) {
            printf("Marks of student %d: ", j+1);
            scanf("%f", &score[i][j]);
            if (score[i][j] > max || score[i][j] < 0) {
                printf("Invalid marks!\n");
                if (j == 0) { j = 10; i--; } else { j--; }
            }
        }
    }
}
```

```

for (int i = 0; i < 5; i++) {
    printf("\n\nSubject %d:\n", i+1);
    for (int j = 0; j < 10; j++) {
        printf("%.2f ", score[i][j]);
    }
}

```

Output:

```

Enter the max marks: 10
Enter the marks for subject 1
Marks of student 1: 1
Marks of student 2: 1
Marks of student 3: 1
Marks of student 4: 1
Marks of student 5: 1
Marks of student 6: 1
Marks of student 7: 1
Marks of student 8: 1
Marks of student 9: 1
Marks of student 10: 1
Enter the marks for subject 2
Marks of student 1: 1
Marks of student 2: 1
Marks of student 3: 1
Marks of student 4: 1
Marks of student 5: 1
Marks of student 6: 1
Marks of student 7: 1
Marks of student 8: 1
Marks of student 9: 1
Marks of student 10: 1
Enter the marks for subject 3
Marks of student 1: 1
Marks of student 2: 1
Marks of student 3: 1
Marks of student 4: 1
Marks of student 5: 1
Marks of student 6: 1
Marks of student 7: 1
Marks of student 8: 1
Marks of student 9: 1
Marks of student 10: 1
Enter the marks for subject 4
Marks of student 1: 1
Marks of student 2: 1
Marks of student 3: 1
Marks of student 4: 1
Marks of student 5: 1
Marks of student 6: 1
Marks of student 7: 1
Marks of student 8: 1
Marks of student 9: 1
Marks of student 10: 1
Enter the marks for subject 5
Marks of student 1: 1
Marks of student 2: 1
Marks of student 3: 1
Marks of student 4: 1
Marks of student 5: 1
Marks of student 6: 1
Marks of student 7: 1
Marks of student 8: 1
Marks of student 9: 1
Marks of student 10: 1
Subject 1:
1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00
Subject 2:
1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00
Subject 3:
1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00
Subject 4:
1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00
Subject 5:
1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00
[Program finished]

```

Q7. Write a program to implement a linked list.

Code:

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

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

struct node *head = NULL;
struct node *current = NULL;

void printList() {
    struct node *ptr = head;
    printf("\n[ ");

    while(ptr != NULL) {
        printf("%d,%d) ",ptr->key,ptr->data);
        ptr = ptr->next;
    }

    printf(" ]");
}

void insertFirst(int key, int data)
{
    struct node *link = (struct node*) malloc(sizeof(struct node));

    link->key = key;
    link->data = data;

    link->next = head;

    head = link;
}

struct node* deleteFirst()
{
    struct node *tempLink = head;

    head = head->next;

    return tempLink;
}

bool isEmpty() {
    return head == NULL;
}

int length() {
    int length = 0;
    struct node *current;

    for(current = head; current != NULL; current = current->next) {
        length++;
    }

    return length;
}
```

```

}

struct node* find(int key) {

    struct node* current = head;

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

    while(current->key != key) {

        if(current->next == NULL) {
            return NULL;
        } else {
            current = current->next;
        }
    }

    return current;
}

struct node* delete(int key) {

    struct node* current = head;
    struct node* previous = NULL;

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

    while(current->key != key) {

        if(current->next == NULL) {
            return NULL;
        } else {
            previous = current;
            current = current->next;
        }
    }

    if(current == head) {
        head = head->next;
    } else {
        previous->next = current->next;
    }

    return current;
}

void sort() {

    int i, j, k, tempKey, tempData;
    struct node *current;
    struct node *next;

    int size = length();
    k = size ;

    for ( i = 0 ; i < size - 1 ; i++, k-- ) {
        current = head;
        next = head->next;

        for ( j = 1 ; j < k ; j++ ) {

```

```

if ( current->data > next->data ) {
    tempData = current->data;
    current->data = next->data;
    next->data = tempData;

    tempKey = current->key;
    current->key = next->key;
    next->key = tempKey;
}

current = current->next;
next = next->next;
}
}
}

void reverse(struct node** head_ref) {
    struct node* prev  = NULL;
    struct node* current = *head_ref;
    struct node* next;

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

    *head_ref = prev;
}

void main() {
    insertFirst(1,10);
    insertFirst(2,20);
    insertFirst(3,30);
    insertFirst(4,1);
    insertFirst(5,40);
    insertFirst(6,56);

    printf("Original List: ");

    printList();

    while(!isEmpty()) {
        struct node *temp = deleteFirst();
        printf("\nDeleted value:");
        printf("(%d,%d) ",temp->key,temp->data);
    }
}

printf("\nList after deleting all items: ");
printList();
insertFirst(1,10);
insertFirst(2,20);
insertFirst(3,30);
insertFirst(4,1);
insertFirst(5,40);
insertFirst(6,56);

printf("\nRestored List: ");
printList();
printf("\n");

struct node *foundLink = find(4);

```

```

if(foundLink != NULL) {
    printf("Element found: ");
    printf("(%d,%d) ",foundLink->key,foundLink->data);
    printf("\n");
} else {
    printf("Element not found.");
}

delete(4);
printf("List after deleting an item: ");
printList();
printf("\n");
foundLink = find(4);

if(foundLink != NULL) {
    printf("Element found: ");
    printf("(%d,%d) ",foundLink->key,foundLink->data);
    printf("\n");
} else {
    printf("Element not found.");
}

printf("\n");
sort();

printf("List after sorting the data: ");
printList();

reverse(&head);
printf("\nList after reversing the data: ");
printList();
}

```

Output:

```

Original List:
[ (6,56) (5,40) (4,1) (3,30) (2,20) (1,10)  ]
Deleted value:(6,56)
Deleted value:(5,40)
Deleted value:(4,1)
Deleted value:(3,30)
Deleted value:(2,20)
Deleted value:(1,10)
List after deleting all items:
[  ]
Restored List:
[ (6,56) (5,40) (4,1) (3,30) (2,20) (1,10)  ]
Element found: (4,1)
List after deleting an item:
[ (6,56) (5,40) (3,30) (2,20) (1,10)  ]
Element not found.
List after sorting the data:
[ (1,10) (2,20) (3,30) (5,40) (6,56)  ]
List after reversing the data:
[ (6,56) (5,40) (3,30) (2,20) (1,10)  ]
[Program finished]

```

Q8. Write a program to insert a node in a linked list and delete a node from a linked list.

Code:

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

```

#include <string.h>
#include <stdlib.h>
#include <stdbool.h>

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

struct node *head = NULL;
struct node *current = NULL;

void printList() {
    struct node *ptr = head;
    printf("\n[ ");

    while(ptr != NULL) {
        printf("%d,%d ",ptr->key,ptr->data);
        ptr = ptr->next;
    }

    printf(" ]");
}

void insertFirst(int key, int data)
{
    struct node *link = (struct node*) malloc(sizeof(struct node));

    link->key = key;
    link->data = data;

    link->next = head;

    head = link;
}

struct node* deleteFirst()
{
    struct node *tempLink = head;

    head = head->next;

    return tempLink;
}

bool isEmpty() {
    return head == NULL;
}

int length() {
    int length = 0;
    struct node *current;

    for(current = head; current != NULL; current = current->next) {
        length++;
    }

    return length;
}

struct node* find(int key) {
    struct node* current = head;

```

```

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

while(current->key != key) {

    if(current->next == NULL) {
        return NULL;
    } else {
        current = current->next;
    }
}

return current;
}

struct node* delete(int key) {

    struct node* current = head;
    struct node* previous = NULL;

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

    while(current->key != key) {

        if(current->next == NULL) {
            return NULL;
        } else {
            previous = current;
            current = current->next;
        }
    }

    if(current == head) {
        head = head->next;
    } else {
        previous->next = current->next;
    }

    return current;
}

void sort() {

    int i, j, k, tempKey, tempData;
    struct node *current;
    struct node *next;

    int size = length();
    k = size ;

    for ( i = 0 ; i < size - 1 ; i++, k- ) {
        current = head;
        next = head->next;

        for ( j = 1 ; j < k ; j++ ) {

            if ( current->data > next->data ) {
                tempData = current->data;
                current->data = next->data;
                next->data = tempData;
            }
        }
    }
}

```

```

        tempKey = current->key;
        current->key = next->key;
        next->key = tempKey;
    }

    current = current->next;
    next = next->next;
}
}

void reverse(struct node** head_ref) {
    struct node* prev  = NULL;
    struct node* current = *head_ref;
    struct node* next;

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

    *head_ref = prev;
}

void main() {
    insertFirst(1,10);
    insertFirst(2,20);
    insertFirst(3,30);
    insertFirst(4,1);
    insertFirst(5,40);
    insertFirst(6,56);

    printf("Original List: ");

    printList();

    while(!isEmpty()) {
        struct node *temp = deleteFirst();
        printf("\nDeleted value:");
        printf("(%d,%d) ",temp->key,temp->data);
    }

    printf("\nList after deleting all items: ");
    printList();
    insertFirst(1,10);
    insertFirst(2,20);
    insertFirst(3,30);
    insertFirst(4,1);
    insertFirst(5,40);
    insertFirst(6,56);

    printf("\nRestored List: ");
    printList();
    printf("\n");

    struct node *foundLink = find(4);

    if(foundLink != NULL) {
        printf("Element found: ");
        printf("(%d,%d) ",foundLink->key,foundLink->data);
        printf("\n");
    }
}
```

```

} else {
    printf("Element not found.");
}

delete(4);
printf("List after deleting an item: ");
printList();
printf("\n");
foundLink = find(4);

if(foundLink != NULL) {
    printf("Element found: ");
    printf("(%d,%d) ",foundLink->key,foundLink->data);
    printf("\n");
} else {
    printf("Element not found.");
}

printf("\n");
sort();

printf("List after sorting the data: ");
printList();

reverse(&head);
printf("\nList after reversing the data: ");
printList();
}

```

Output:

```

Original List:
[ (6,56) (5,40) (4,1) (3,30) (2,20) (1,10)  ]
Deleted value:(6,56)
Deleted value:(5,40)
Deleted value:(4,1)
Deleted value:(3,30)
Deleted value:(2,20)
Deleted value:(1,10)
List after deleting all items:
[  ]
Restored List:
[ (6,56) (5,40) (4,1) (3,30) (2,20) (1,10)  ]
Element found: (4,1)
List after deleting an item:
[ (6,56) (5,40) (3,30) (2,20) (1,10)  ]
Element not found.
List after sorting the data:
[ (1,10) (2,20) (3,30) (5,40) (6,56)  ]
List after reversing the data:
[ (6,56) (5,40) (3,30) (2,20) (1,10)  ]
[Program finished]

```

Q9. Write a program to print the elements of a linked list in reverse order without disturbing the linked list.

Code:

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

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

void printReverse(struct Node* head)
{
    if (head == NULL)
        return;

    printReverse(head->next);

    printf("%d ", head->data);
}

void push(struct Node** head_ref, char new_data)
{
    struct Node* new_node =
        (struct Node*) malloc(sizeof(struct Node));

    new_node->data = new_data;

    new_node->next = (*head_ref);

    (*head_ref) = new_node;
}

int main()
{
    struct Node* head = NULL;
    push(&head, 4);
    push(&head, 3);
    push(&head, 2);
    push(&head, 1);
    printReverse(head);
    return 0;
}
```

Output:

```
4 3 2 1
[Program finished]
```

Q10. Write a program to reverse a linked list.

Code:

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

```
struct Node
{
```

```

int data;
struct Node* next;
};

static void reverse(struct Node** head_ref)
{
    struct Node* prev = NULL;
    struct Node* current = *head_ref;
    struct Node* next;
    while (current != NULL)
    {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    *head_ref = prev;
}

void push(struct Node** head_ref, int new_data)
{
    struct Node* new_node =
        (struct Node*) malloc(sizeof(struct Node));

    new_node->data = new_data;

    new_node->next = (*head_ref);

    (*head_ref) = new_node;
}

void printList(struct Node *head)
{
    struct Node *temp = head;
    while(temp != NULL)
    {
        printf("%d ", temp->data);
        temp = temp->next;
    }
}

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

    push(&head, 20);
    push(&head, 4);
    push(&head, 15);
    push(&head, 85);

    printf("Given linked list\n");
    printList(head);
    reverse(&head);
    printf("\nReversed Linked list \n");
    printList(head);
    getchar();
}

```

```

Given linked list
85 15 4 20
Reversed Linked list
20 4 15 85 █

```

Output:

Q11. Write a program to add two polynomials using linked lists.

Code:

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

typedef struct link {
    int coeff;
    int pow;
    struct link * next;
} my_poly;

void my_create_poly(my_poly **);
void my_show_poly(my_poly *);
void my_add_poly(my_poly **, my_poly *, my_poly *);

int main(void) {
    int ch;
    do {
        my_poly * poly1, * poly2, * poly3;

        printf("\nCreate 1st expression\n");
        my_create_poly(&poly1);
        printf("\nStored the 1st expression");
        my_show_poly(poly1);

        printf("\nCreate 2nd expression\n");
        my_create_poly(&poly2);
        printf("\nStored the 2nd expression");
        my_show_poly(poly2);

        my_add_poly(&poly3, poly1, poly2);
        my_show_poly(poly3);

        printf("\nAdd two more expressions? (Y = 1/N = 0): ");
        scanf("%d", &ch);
    } while (ch);
    return 0;
}

void my_create_poly(my_poly ** node) {
    int flag;
    int coeff, pow;
    my_poly * tmp_node;
    tmp_node = (my_poly *) malloc(sizeof(my_poly));
    *node = tmp_node;
    do {
        printf("\nEnter Coeff:");
        scanf("%d", &coeff);
        tmp_node->coeff = coeff;
        printf("\nEnter Pow:");
        scanf("%d", &pow);
        tmp_node->pow = pow;

        tmp_node->next = NULL;

        printf("\nContinue adding more terms to the polynomial list?(Y = 1/N = 0): ");
        scanf("%d", &flag);
        printf("\nFLAG: %c\n", flag);
        if(flag) {
            tmp_node->next = (my_poly *) malloc(sizeof(my_poly));
            tmp_node = tmp_node->next;
            tmp_node->next = NULL;
        }
    } while (flag);
}
```

```

}

void my_show_poly(my_poly * node) {
    printf("\nThe polynomial expression is:\n");
    while(node != NULL) {
        printf("%dx^%d", node->coeff, node->pow);
        node = node->next;
        if(node != NULL)
            printf(" + ");
    }
}

void my_add_poly(my_poly ** result, my_poly * poly1, my_poly * poly2) {
    my_poly * tmp_node;
    tmp_node = (my_poly *) malloc(sizeof(my_poly));
    tmp_node->next = NULL;
    *result = tmp_node;

    while(poly1 && poly2) {
        if (poly1->pow > poly2->pow) {
            tmp_node->pow = poly1->pow;
            tmp_node->coeff = poly1->coeff;
            poly1 = poly1->next;
        }
        else if (poly1->pow < poly2->pow) {
            tmp_node->pow = poly2->pow;
            tmp_node->coeff = poly2->coeff;
            poly2 = poly2->next;
        }
        else {
            tmp_node->pow = poly1->pow;
            tmp_node->coeff = poly1->coeff + poly2->coeff;
            poly1 = poly1->next;
            poly2 = poly2->next;
        }

        if(poly1 && poly2) {
            tmp_node->next = (my_poly *) malloc(sizeof(my_poly));
            tmp_node = tmp_node->next;
            tmp_node->next = NULL;
        }
    }

    while(poly1 || poly2)
    {
        tmp_node->next = (my_poly *) malloc(sizeof(my_poly));
        tmp_node = tmp_node->next;
        tmp_node->next = NULL;

        if(poly1) {
            tmp_node->pow = poly1->pow;
            tmp_node->coeff = poly1->coeff;
            poly1 = poly1->next;
        }
        if(poly2) {
            tmp_node->pow = poly2->pow;
            tmp_node->coeff = poly2->coeff;
            poly2 = poly2->next;
        }
    }

    printf("\nAddition Complete");
}

```

Output:

```
Create 1st expression

Enter Coeff:2

Enter Pow:2

Continue adding more terms to the polynomial list?(Y = 1/N = 0)
: 1

FLAG:

Enter Coeff:1

Enter Pow:1

Continue adding more terms to the polynomial list?(Y = 1/N = 0)
: 1

FLAG:

Enter Coeff:1

Enter Pow:0

Continue adding more terms to the polynomial list?(Y = 1/N = 0)
: 0

FLAG:

Stored the 1st expression
The polynomial expression is:
2x^2 + 1x^1 + 1x^0
Create 2nd expression

Enter Coeff:2

Enter Pow:2

Continue adding more terms to the polynomial list?(Y = 1/N = 0)
: 1

Continue adding more terms to the polynomial list?(Y = 1/N = 0)
: 0

FLAG:

Stored the 2nd expression
The polynomial expression is:
2x^2 + 1x^1 + 1x^0
Addition Complete
The polynomial expression is:
4x^2 + 2x^1 + 2x^0
Add two more expressions? (Y = 1/N = 0): 0

[Program finished]
```

Q12. Write a program to implement a doubly-linked list.

Code:

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

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

struct node *head = NULL;
struct node *last = NULL;
struct node *current = NULL;

bool isEmpty() {
    return head == NULL;
}

int length() {
    int length = 0;
    struct node *current;

    for(current = head; current != NULL; current = current->next){
        length++;
    }

    return length;
}
```

```
void displayForward() {

    struct node *ptr = head;

    printf("\n[ ");

    while(ptr != NULL) {
        printf("(%d,%d) ",ptr->key,ptr->data);
        ptr = ptr->next;
    }

    printf(" ]");
}

void displayBackward() {

    struct node *ptr = last;

    printf("\n[ ");

    while(ptr != NULL) {
        printf("(%d,%d) ",ptr->key,ptr->data);

        ptr = ptr ->prev;
    }

}

void insertFirst(int key, int data) {

    struct node *link = (struct node*) malloc(sizeof(struct node));
    link->key = key;
```

```
link->data = data;

if(isEmpty()) {
    last = link;
} else {
    head->prev = link;
}

link->next = head;

head = link;
}

void insertLast(int key, int data) {

struct node *link = (struct node*) malloc(sizeof(struct node));
link->key = key;
link->data = data;

if(isEmpty()) {
    last = link;
} else {
    last->next = link;

    link->prev = last;
}

last = link;
}

struct node* deleteFirst() {

struct node *tempLink = head;

if(head->next == NULL){
```

```
last = NULL;  
} else {  
    head->next->prev = NULL;  
}  
  
head = head->next;  
return tempLink;  
}
```

```
struct node* deleteLast() {  
    struct node *tempLink = last;  
  
    if(head->next == NULL) {  
        head = NULL;  
    } else {  
        last->prev->next = NULL;  
    }  
  
    last = last->prev;  
  
    return tempLink;  
}
```

```
struct node* delete(int key) {  
  
    struct node* current = head;  
    struct node* previous = NULL;  
  
    if(head == NULL) {  
        return NULL;  
    }  
  
    while(current->key != key) {
```

```

if(current->next == NULL) {
    return NULL;
} else {
    previous = current;

    current = current->next;
}

}

if(current == head) {
    head = head->next;
} else {
    current->prev->next = current->next;
}

if(current == last) {
    last = current->prev;
} else {
    current->next->prev = current->prev;
}

return current;
}

bool insertAfter(int key, int newKey, int data) {
    struct node *current = head;

    if(head == NULL) {
        return false;
    }

    while(current->key != key) {

        if(current->next == NULL) {

```

```

    return false;
} else {
    current = current->next;
}
}

struct node *newLink = (struct node*) malloc(sizeof(struct node));
newLink->key = newKey;
newLink->data = data;

if(current == last) {
    newLink->next = NULL;
    last = newLink;
} else {
    newLink->next = current->next;
    current->next->prev = newLink;
}

newLink->prev = current;
current->next = newLink;
return true;
}

void main() {
    insertFirst(1,10);
    insertFirst(2,20);
    insertFirst(3,30);
    insertFirst(4,1);
    insertFirst(5,40);
    insertFirst(6,56);

    printf("\nList (First to Last): ");
    displayForward();

    printf("\n");
}

```

```

printf("\nList (Last to first): ");
displayBackward();

printf("\nList , after deleting first record: ");
deleteFirst();
displayForward();

printf("\nList , after deleting last record: ");
deleteLast();
displayForward();

printf("\nList , insert after key(4) : ");
insertAfter(4,7, 13);
displayForward();

printf("\nList , after delete key(4) : ");
delete(4);
displayForward();
}

```

Output:

```

List (First to Last):
[ (6,56) (5,40) (4,1) (3,30) (2,20) (1,10)  ]

List (Last to first):
[ (1,10) (2,20) (3,30) (4,1) (5,40) (6,56)
List , after deleting first record:
[ (5,40) (4,1) (3,30) (2,20) (1,10)  ]
List , after deleting last record:
[ (5,40) (4,1) (3,30) (2,20)  ]
List , insert after key(4) :
[ (5,40) (4,1) (7,13) (3,30) (2,20)  ]
List , after delete key(4) :
[ (5,40) (7,13) (3,30) (2,20)  ]
[Program finished]

```

Q13. Write a program to implement a stack using an array.

Code:

```
#include <stdio.h>
#define SIZE 3
```

```
int arr[SIZE], top = -1;

void peek() {
    top == -1 ? printf("Stack is empty!\n") : printf("%d\n", arr[top]);
}

void push(int val) {
    if (top == SIZE - 1) {
        printf("Overflow!\n");
    } else {
        arr[++top] = val;
        printf("Successfully pushed %d!\n", val);
    }
}

int pop() {
    if (top == -1) {
        printf("Underflow!\n");
    } else {
        int temp = arr[top--];
        return temp;
    }
}

int main() {
    peek();

    push(3);
    push(4);
    push(5);
    push(6);

    peek();
```

```
printf("Popped %d", pop());
printf("\nPopped %d", pop());
printf("\nPopped %d\n", pop());
pop();
}
```

Output:

```
Stack is empty!
Successfully pushed 3!
Successfully pushed 4!
Successfully pushed 5!
Overflow!
5
Popped 5
Popped 4
Popped 3
Underflow!
```

```
[Program finished]
```

Q14. Write a program to implement a stack using a linked list.

Code:

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

```
struct node
{
    int info;
    struct node *ptr;
}*top,*top1,*temp;
```

```
int topelement();
void push(int data);
void pop();
void empty();
void display();
void destroy();
void stack_count();
void create();
```

```
int count = 0;

void main()
{
    int no, ch, e;

    printf("\n 1 - Push");
    printf("\n 2 - Pop");
    printf("\n 3 - Top");
    printf("\n 4 - Empty");
    printf("\n 5 - Exit");
    printf("\n 6 - Dipslay");
    printf("\n 7 - Stack Count");
    printf("\n 8 - Destroy stack");

    create();

    while (1)
    {
        printf("\n Enter choice : ");
        scanf("%d", &ch);

        switch (ch)
        {
            case 1:
                printf("Enter data : ");
                scanf("%d", &no);
                push(no);
                break;
            case 2:
                pop();
                break;
            case 3:
                if (top == NULL)
```

```

        printf("No elements in stack");
    else
    {
        e = topelement();
        printf("\n Top element : %d", e);
    }
    break;
case 4:
    empty();
    break;
case 5:
    exit(0);
case 6:
    display();
    break;
case 7:
    stack_count();
    break;
case 8:
    destroy();
    break;
default :
    printf(" Wrong choice, Please enter correct choice ");
    break;
}
}
}

```

void create()

```
{
    top = NULL;
}
```

void stack_count()

```
{
```

```

printf("\n No. of elements in stack : %d", count);
}

void push(int data)
{
    if (top == NULL)
    {
        top =(struct node *)malloc(1*sizeof(struct node));
        top->ptr = NULL;
        top->info = data;
    }
    else
    {
        temp =(struct node *)malloc(1*sizeof(struct node));
        temp->ptr = top;
        temp->info = data;
        top = temp;
    }
    count++;
}

void display()
{
    top1 = top;

    if (top1 == NULL)
    {
        printf("Stack is empty");
        return;
    }

    while (top1 != NULL)
    {
        printf("%d ", top1->info);
        top1 = top1->ptr;
    }
}

```

```

        }

    }

void pop()
{
    top1 = top;

    if (top1 == NULL)
    {
        printf("\n Error : Trying to pop from empty stack");
        return;
    }
    else
        top1 = top1->ptr;
    printf("\n Popped value : %d", top->info);
    free(top);
    top = top1;
    count--;
}

int topelement()
{
    return(top->info);
}

void empty()
{
    if (top == NULL)
        printf("\n Stack is empty");
    else
        printf("\n Stack is not empty with %d elements", count);
}

void destroy()
{

```

```
top1 = top;

while (top1 != NULL)
{
    top1 = top->ptr;
    free(top);
    top = top1;
    top1 = top1->ptr;
}

free(top1);
top = NULL;

printf("\n All stack elements destroyed");
count = 0;
```

Output:

```
1 - Push
2 - Pop
3 - Top
4 - Empty
5 - Exit
6 - Display
7 - Stack Count
8 - Destroy stack
Enter choice : 8
```

```
All stack elements destroyed
Enter choice : 7
```

```
No. of elements in stack : 0
Enter choice : 6
Stack is empty
Enter choice : 4
```

```
Stack is empty
Enter choice : 3
No elements in stack
Enter choice : 2
```

```
Error : Trying to pop from empty stack
Enter choice : 1
Enter data : 1
```

```
Enter choice : 6
1
Enter choice : 5
```

```
[Program finished]
```

Q15. Write a program to implement a queue using an array.

Code:

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

int queue[MAX];
int f = -1, r = -1, size = -1;

void enqueue(int val) {
    if(size < MAX) {
        if (size < 0) {
            queue[0] = val;
            f++; r++;
            size = 1;
        } else if (r == MAX-1) {
            queue[0] = val;
            r = 0;
            size++;
        } else {
            queue[++r] = val;
            size++;
        }
    } else {
        printf("Queue is full\n");
    }
}

int dequeue() {
    if (size < 0) {
        printf("Queue is empty\n");
    } else {
        size--;
        f++;
    }
}
```

```
void display()
{
    int i;
    if( r >= f ) {
        for (i = f; i <= r; i++) {
            printf("%d ",queue[i]);
        }
    } else {
        for (i = f; i < MAX; i++) {
            printf("%d ",queue[i]);
        }
        for (i = 0; i <= r; i++) {
            printf("%d ",queue[i]);
        }
    }
}
```

```
int main()
{
    enqueue(24);
    enqueue(9);
    enqueue(22);
    enqueue(93);
    display();
    dequeue();
    printf("\nAfter dequeue\n");
    display();
    enqueue(8);
    enqueue(63);
    enqueue(57);
    enqueue(900);
    dequeue();
    enqueue(84);
    enqueue(73);
```

```
printf("\nAfter enqueue\n");
display();
return 0;
}
```

Output:

```
24 9 22 93
After dequeue
9 22 93
After enqueue
22 93 8 63 57 900 84 73
[Program finished]
```

Q16. Write a program to implement a queue using a linked list.

Code:

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

struct node
{
    int info;
    struct node *ptr;
}*front,*rear,*temp,*front1;
```

```
int frontelement();
```

```
void enq(int data);
```

```
void deq();
```

```
void empty();
```

```
void display();
```

```
void create();
```

```
void queuesize();
```

```
int count = 0;
```

```
void main()
```

```
{
```

```

int no, ch, e;

printf("\n 1 - Enque");
printf("\n 2 - Deque");
printf("\n 3 - Front element");
printf("\n 4 - Empty");
printf("\n 5 - Exit");
printf("\n 6 - Display");
printf("\n 7 - Queue size");
create();
while (1)
{
    printf("\n Enter choice : ");
    scanf("%d", &ch);
    switch (ch)
    {
        case 1:
            printf("Enter data : ");
            scanf("%d", &no);
            enq(no);
            break;
        case 2:
            deq();
            break;
        case 3:
            e = frontelement();
            if (e != 0)
                printf("Front element : %d", e);
            else
                printf("\n No front element in Queue as queue is empty");
            break;
        case 4:
            empty();
            break;
        case 5:

```

```

    exit(0);

case 6:
    display();
    break;

case 7:
    queuesize();
    break;

default:
    printf("Wrong choice, Please enter correct choice ");
    break;
}
}
}

```

```

void create()
{
    front = rear = NULL;
}
```

```

void queuesize()
{
    printf("\n Queue size : %d", count);
}
```

```

void enq(int data)
{
    if (rear == NULL)
    {
        rear = (struct node *)malloc(1*sizeof(struct node));
        rear->ptr = NULL;
        rear->info = data;
        front = rear;
    }
    else
    {
```

```

temp=(struct node *)malloc(1*sizeof(struct node));
rear->ptr = temp;
temp->info = data;
temp->ptr = NULL;

rear = temp;
}

count++;

}

void display()
{
front1 = front;

if ((front1 == NULL) && (rear == NULL))
{
printf("Queue is empty");
return;
}

while (front1 != rear)
{
printf("%d ", front1->info);
front1 = front1->ptr;
}

if (front1 == rear)
printf("%d", front1->info);
}

void deq()
{
front1 = front;

if (front1 == NULL)
{
printf("\n Error: Trying to display elements from empty queue");
}

```

```

    return;
}

else
if (front1->ptr != NULL)
{
    front1 = front1->ptr;
    printf("\n Dequeued value : %d", front->info);
    free(front);
    front = front1;
}
else
{
    printf("\n Dequeued value : %d", front->info);
    free(front);
    front = NULL;
    rear = NULL;
}
count--;
}

```

```

int frontelement()
{
    if ((front != NULL) && (rear != NULL))
        return(front->info);
    else
        return 0;
}

```

```

void empty()
{
    if ((front == NULL) && (rear == NULL))
        printf("\n Queue empty");
    else
        printf("Queue not empty");
}

```

Output:

```
1 - Enque
2 - Deque
3 - Front element
4 - Empty
5 - Exit
6 - Display
7 - Queue size
Enter choice : 7

Queue size : 0
Enter choice : 6
Queue is empty
Enter choice : 4

Queue empty
Enter choice : 3

No front element in Queue as queue is empty
Enter choice : 2

Error: Trying to display elements from empty queue
Enter choice : 1
Enter data : 1

Enter choice : 6
1
Enter choice : 5

[Program finished]
```

Q17. Write a program to implement a circular queue using an array.

Code:

```
#include <stdio.h>

#define SIZE 5

int cirqueue[SIZE], f = -1, r = -1;

int isFull() {
    if ((f == r + 1) || (f == 0 && r == SIZE - 1)) { return 1; }
    return 0;
}

int isEmpty() {
    if (f == -1) { return 1; }
    return 0;
```

```
}
```

```
void enqueue(int val) {  
    if (isFull()) { printf("\nQueue is full!\n"); }  
    else {  
        if (f == -1) { f = 0; }  
        r = (r + 1) % SIZE;  
        cirqueue[r] = val;  
        printf("\nInserted: %d", val);  
    }  
}
```

```
int dequeue() {  
    int val;  
    if (isEmpty()) {  
        printf("\nQueue is empty!\n");  
    } else {  
        val = cirqueue[f];  
        if (f == r) { f = r = -1; }  
        else {  
            f = (f + 1) % SIZE;  
        } printf("\nDeleted element: %d \n", val);  
        return val;  
    }  
}
```

```
void display() {  
    int i;  
    if (isEmpty())  
        printf("\nEmpty Queue\n");  
    else {  
        printf("\nFront: %d ", f);  
        printf("\nCircular Queue: ");  
        for (i = f; i != r; i = (i + 1) % SIZE) {  
            printf("%d ", cirqueue[i]);  
        }  
    }  
}
```

```

    }

    printf("%d ", cirqueue[i]);
    printf("\nrear: %d \n", r);
}

}

int main() {
    dequeue();

    enqueue(1);
    enqueue(2);
    enqueue(3);
    enqueue(4);
    enqueue(5);

    enqueue(6);

    display();
    dequeue();

    display();

    enqueue(7);
    display();

    enqueue(8);

    return 0;
}

```

Output:

```

Queue is empty!
Inserted: 1
Inserted: 2
Inserted: 3
Inserted: 4
Inserted: 5
Queue is full!
Front: 0
Circular Queue: 1 2 3 4 5
rear: 4
Deleted element: 1
Front: 1
Circular Queue: 2 3 4 5
rear: 4
Inserted: 7
Front: 1
Circular Queue: 2 3 4 5 7
rear: 0
Queue is full!
[Program finished]■

```

Q18. Write a program to implement a priority queue using a linked list.

Code:

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

typedef struct node {
    int val;
    int priority;
    struct node* next;
} Node;

Node* initNode(int v, int p) {
    Node* temp = (Node*) malloc( sizeof(Node) );
    temp -> val = v;
    temp -> priority = p;
    temp -> next = NULL;
    return temp;
}

int peek(Node** head) {
    return (*head) -> val;
}

void pop(Node** head) {
    Node* temp = *head;
    (*head) = (*head) -> next;
    free(temp);
}

void push(Node** head, int v, int p) {
    Node* first = (*head);
    Node* temp = initNode(v, p);
    if ((*head) -> priority > p) {
        temp -> next = *head;
        (*head) = temp;
    } else {
        while (first -> next != NULL && first -> next -> priority < p) {
            first = first -> next;
        }
        temp -> next = first -> next;
        first -> next = temp;
    }
}
```

```

    }

    temp -> next = first -> next;
    first -> next = temp;
} printf("Successfully pushed %d!\n", v);
}

int isEmpty(Node** head) {
    return (*head) == NULL;
}

int main() {

    Node* pq = initNode(7, 1);

    printf("Created linked list with value 7.\n");
    push(&pq, 1, 2);
    push(&pq, 3, 3);
    push(&pq, 2, 0);

    while (!isEmpty(&pq)) {
        printf("Popped %d\n", peek(&pq));
        pop(&pq);
    }

    printf("Priority queue is now empty.");
    return 0;
}

```

Output:

```

Created linked list with value 7.
Successfully pushed 1!
Successfully pushed 3!
Successfully pushed 2!
Popped 2
Popped 7
Popped 1
Popped 3
Priority queue is now empty.
[Program finished]

```

Q19. Write a program to implement a double-ended queue using a linked list.

Code:

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

```
typedef struct node {
```

```
    int data;
```

```

struct node *prev, *next;
} Node;

Node *head = NULL, *end = NULL;

Node* initNode(int data) {
    Node *new = (Node *) malloc(sizeof (Node));
    new -> data = data;
    new -> next = new -> prev = NULL;
    return new;
}

void makeEnds() {
    head = initNode(0);
    end = initNode(0);
    head -> next = end;
    end -> prev = head;
}

void enqueueFront(int data) {
    Node *new, *temp;
    new = initNode(data);
    temp = head -> next;
    head -> next = new;
    new -> prev = head;
    new -> next = temp;
    temp -> prev = new;
}

void enqueueRear(int data) {
    Node *new, *temp;
    new = initNode(data);
    temp = end -> prev;
    end -> prev = new;
    new -> next = end;
}

```

```
    new -> prev = temp;  
    temp -> next = new;  
}
```

```
void dequeueFront() {  
    Node *temp;  
    if (head -> next == end) {  
        printf("Queue is empty\n");  
    } else {  
        temp = head -> next;  
        head -> next = temp -> next;  
        temp -> next -> prev = head;  
        free(temp);  
    } return;  
}
```

```
void dequeueRear() {  
    Node *temp;  
    if (end -> prev == head) {  
        printf("Queue is empty\n");  
    } else {  
        temp = end -> prev;  
        end -> prev = temp -> prev;  
        temp -> prev -> next = end;  
        free(temp);  
    } return;  
}
```

```
void display() {  
    Node *temp;  
  
    if (head -> next == end) {  
        printf("Queue is empty\n");  
        return;  
    }
```

```
temp = head -> next;  
while (temp != end) {  
    printf("%-3d", temp -> data);  
    temp = temp -> next;  
}  
printf("\n");  
}
```

```
int main() {  
    makeEnds();  
  
    enqueueFront(23);  
    enqueueRear(29);  
    enqueueRear(30);  
    enqueueFront(40);  
    display();  
  
    dequeueFront();  
    dequeueRear();  
    display();  
  
    dequeueFront();  
    dequeueRear();  
    display();  
    return 0;  
}
```

Output:

```
40 23 29 30  
23 29  
Queue is empty  
[Program finished]
```

Q20. Write a program to construct a binary tree and display its preorder, inorder and postorder traversals.

Code:

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

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

Node* newNode(int data) {
    Node* node = (Node*) malloc(sizeof(Node));
    node->data = data;
    node->left = NULL;
    node->right = NULL;
    return node;
}

void Postorder(Node* node) {
    if (node == NULL) { return; }

    Postorder(node->left);
    Postorder(node->right);

    printf("%d ", node->data);
}

void Inorder(Node* node)
{
    if (node == NULL) { return; }

    Inorder(node->left);
```

```
printf("%d ", node -> data);

Inorder(node -> right);

}

void Preorder(Node* node)
{
    if (node == NULL) { return; }

    printf("%d ", node -> data);

    Preorder(node -> left);

    Preorder(node -> right);
}

int main()
{
    Node* base = newNode(1);
    base -> left = newNode(2);
    base -> right = newNode(3);
    base -> left -> left = newNode(4);
    base -> left -> right = newNode(5);
    base -> right -> left = newNode(6);
    base -> right -> right = newNode(7);

    printf("\nPreorder traversal of binary tree:\n");
    Preorder(base);

    printf("\nInorder traversal of binary tree:\n");
    Inorder(base);

    printf("\nPostorder traversal of binary tree:\n");
    Postorder(base);

    return 0;
}
```

```
}
```

Output:

```
Preorder traversal of binary tree:  
1 2 4 5 3 6 7  
Inorder traversal of binary tree:  
4 2 5 1 6 3 7  
Postorder traversal of binary tree:  
4 5 2 6 7 3 1  
[Program finished]
```

Q21. Write a program to construct a binary search tree.

Code:

```
#include <stdio.h>  
#include <stdlib.h>  
  
struct btnode  
{  
    int value;  
    struct btnode *l;  
    struct btnode *r;  
}*root = NULL, *temp = NULL, *t2, *t1;  
  
void delete1();  
void insert();  
void delete();  
void inorder(struct btnode *t);  
void create();  
void search(struct btnode *t);  
void preorder(struct btnode *t);  
void postorder(struct btnode *t);  
void search1(struct btnode *t,int data);  
int smallest(struct btnode *t);  
int largest(struct btnode *t);  
  
int flag = 1;  
  
void main()  
{  
    int ch;  
  
    printf("\nOPERATIONS --");  
    printf("\n1 - Insert an element into tree\n");  
    printf("2 - Delete an element from the tree\n");  
    printf("3 - Inorder Traversal\n");  
    printf("4 - Preorder Traversal\n");  
    printf("5 - Postorder Traversal\n");  
    printf("6 - Exit\n");  
    while(1)  
    {  
        printf("\nEnter your choice : ");  
        scanf("%d", &ch);  
        switch (ch)  
        {  
        case 1:  
            insert();  
            break;
```

```

        case 2:
            delete();
            break;
        case 3:
            inorder(root);
            break;
        case 4:
            preorder(root);
            break;
        case 5:
            postorder(root);
            break;
        case 6:
            exit(0);
        default :
            printf("Wrong choice, Please enter correct choice ");
            break;
    }
}

void insert()
{
    create();
    if (root == NULL)
        root = temp;
    else
        search(root);
}

void create()
{
    int data;

    printf("Enter data of node to be inserted : ");
    scanf("%d", &data);
    temp = (struct btnode *)malloc(1*sizeof(struct btnode));
    temp->value = data;
    temp->l = temp->r = NULL;
}

void search(struct btnode *t)
{
    if ((temp->value > t->value) && (t->r != NULL))
        search(t->r);
    else if ((temp->value > t->value) && (t->r == NULL))
        t->r = temp;
    else if ((temp->value < t->value) && (t->l != NULL))
        search(t->l);
    else if ((temp->value < t->value) && (t->l == NULL))
        t->l = temp;
}

void inorder(struct btnode *t)
{
    if (root == NULL)
    {
        printf("No elements in a tree to display");
        return;
    }
    if (t->l != NULL)
        inorder(t->l);
    printf("%d -> ", t->value);
    if (t->r != NULL)

```

```

        inorder(t->r);
    }

void delete()
{
    int data;

    if (root == NULL)
    {
        printf("No elements in a tree to delete");
        return;
    }
    printf("Enter the data to be deleted : ");
    scanf("%d", &data);
    t1 = root;
    t2 = root;
    search1(root, data);
}

void preorder(struct btnode *t)
{
    if (root == NULL)
    {
        printf("No elements in a tree to display");
        return;
    }
    printf("%d -> ", t->value);
    if (t->l != NULL)
        preorder(t->l);
    if (t->r != NULL)
        preorder(t->r);
}

void postorder(struct btnode *t)
{
    if (root == NULL)
    {
        printf("No elements in a tree to display ");
        return;
    }
    if (t->l != NULL)
        postorder(t->l);
    if (t->r != NULL)
        postorder(t->r);
    printf("%d -> ", t->value);
}

void search1(struct btnode *t, int data)
{
    if ((data>t->value))
    {
        t1 = t;
        search1(t->r, data);
    }
    else if ((data < t->value))
    {
        t1 = t;
        search1(t->l, data);
    }
    else if ((data==t->value))
    {
        delete1(t);
    }
}

```

```

void delete1(struct btnode *t)
{
    int k;

    if ((t->l == NULL) && (t->r == NULL))
    {
        if (t1->l == t)
        {
            t1->l = NULL;
        }
        else
        {
            t1->r = NULL;
        }
        t = NULL;
        free(t);
        return;
    }

    else if ((t->r == NULL))
    {
        if (t1 == t)
        {
            root = t->l;
            t1 = root;
        }
        else if (t1->l == t)
        {
            t1->l = t->l;

        }
        else
        {
            t1->r = t->l;
        }
        t = NULL;
        free(t);
        return;
    }

    else if (t->l == NULL)
    {
        if (t1 == t)
        {
            root = t->r;
            t1 = root;
        }
        else if (t1->r == t)
            t1->r = t->r;
        else
            t1->l = t->r;
        t == NULL;
        free(t);
        return;
    }

    else if ((t->l != NULL) && (t->r != NULL))
    {
        t2 = root;
        if (t->r != NULL)
        {
            k = smallest(t->r);
            flag = 1;

```

```

        }
    else
    {
        k =largest(t->l);
        flag = 2;
    }
    search1(root, k);
    t->value = k;
}

int smallest(struct btnode *t)
{
    t2 = t;
    if (t->l != NULL)
    {
        t2 = t;
        return(smallest(t->l));
    }
    else
        return (t->value);
}

int largest(struct btnode *t)
{
    if (t->r != NULL)
    {
        t2 = t;
        return(largest(t->r));
    }
    else
        return(t->value);
}

```

Output:

```

OPERATIONS ---
1 - Insert an element into tree
2 - Delete an element from the tree
3 - Inorder Traversal
4 - Preorder Traversal
5 - Postorder Traversal
6 - Exit

Enter your choice : 2
No elements in a tree to delete
Enter your choice : 3
No elements in a tree to display
Enter your choice : 4
No elements in a tree to display
Enter your choice : 5
No elements in a tree to display
Enter your choice : 1
Enter data of node to be inserted : 1

Enter your choice : 3
1 ->
Enter your choice : 6

[Program finished]

```

22. Write a program to construct a tree.

```
#include <bits/stdc++.h>
using namespace std;

// A utility function to add an edge in an
// undirected graph.
void addEdge(vector<int> adj[], int u, int v)
{
    adj[u].push_back(v);
    adj[v].push_back(u);
}

// A utility function to print the adjacency list
// representation of graph
void printGraph(vector<int> adj[], int V)
{
    for (int v = 0; v < V; ++v)
    {
        cout << "\n Adjacency list of vertex " << v
             << "\n head ";
        for (auto x : adj[v])
            cout << "-> " << x;
        printf("\n");
    }
}

// Driver code
int main()
{
    int V = 5;
    vector<int> adj[V];
    addEdge(adj, 0, 1);
    addEdge(adj, 0, 4);
    addEdge(adj, 1, 2);
    addEdge(adj, 1, 3);
```

```
    addEdge(adj, 1, 4);
    addEdge(adj, 2, 3);
    addEdge(adj, 3, 4);
    printGraph(adj, V);
    return 0;
}
```

Output :

```
Adjacency list of vertex 0
head -> 1-> 4

Adjacency list of vertex 1
head -> 0-> 2-> 3-> 4

Adjacency list of vertex 2
head -> 1-> 3

Adjacency list of vertex 3
head -> 1-> 2-> 4

Adjacency list of vertex 4
head -> 0-> 1-> 3
```

23. Write a program to calculate the distance between two vertices in a graph

code:

```
#include <bits/stdc++.h>

using namespace std;

int minEdgeBFS(vector<int> edges[], int u,
               int v, int n)

{
    vector<bool> visited(n, 0);
    vector<int> distance(n, 0);
    queue<int> Q;
    distance[u] = 0;
    Q.push(u);
    visited[u] = true;
    while (!Q.empty())
```

```

{
    int x = Q.front();
    Q.pop();
    for (int i = 0; i < edges[x].size(); i++)
    {
        if (visited[edges[x][i]])
            continue;
        distance[edges[x][i]] = distance[x] + 1;
        Q.push(edges[x][i]);
        visited[edges[x][i]] = 1;
    }
}
return distance[v];
}

void addEdge(vector<int> edges[], int u, int v)
{
    edges[u].push_back(v);
    edges[v].push_back(u);
}

int main()
{
    int n = 9;
    vector<int> edges[9];
    addEdge(edges, 0, 1);
    addEdge(edges, 0, 7);
    addEdge(edges, 1, 7);
    addEdge(edges, 1, 2);
    addEdge(edges, 2, 3);
    addEdge(edges, 2, 5);
    addEdge(edges, 2, 8);
    addEdge(edges, 3, 4);
    addEdge(edges, 3, 5);
    addEdge(edges, 4, 5);
    addEdge(edges, 5, 6);
    addEdge(edges, 6, 7);
}

```

```

addEdge(edges, 7, 8);

int u = 0;

int v = 5;

cout << minEdgeBFS(edges, u, v, n) << endl;

return 0;

}

```

Output :

```

laksshaysehrawat@Laksshays-MacBook-Air:~/Desktop/C++/file/"graph"
3

```

24. Write a program to calculate the distances between every pairs of vertices in a graph

code :

```

#include <stdio.h>

#define nV 4

#define INF 999

void printMatrix(int matrix[][nV]);

void floydWarshall(int graph[][nV])

{

    int matrix[nV][nV], i, j, k;

    for (i = 0; i < nV; i++)

        for (j = 0; j < nV; j++)

            matrix[i][j] = graph[i][j];

    for (k = 0; k < nV; k++)

    {

        for (i = 0; i < nV; i++)

        {

            for (j = 0; j < nV; j++)

            {

                if (matrix[i][k] + matrix[k][j] < matrix[i][j])

                    matrix[i][j] = matrix[i][k] + matrix[k][j];

            }

        }

    }

    printMatrix(matrix);
}

```

```

}

void printMatrix(int matrix[][nV])
{
    for (int i = 0; i < nV; i++)
    {
        for (int j = 0; j < nV; j++)
        {
            if (matrix[i][j] == INF)
                printf("%4s", "INF");
            else
                printf("%4d", matrix[i][j]);
        }
        printf("\n");
    }
}

int main()
{
    int graph[nV][nV] = {{0, 3, INF, 5},
                         {2, 0, INF, 4},
                         {INF, 1, 0, INF},
                         {INF, INF, 2, 0}};

    floydWarshall(graph);
}

```

Output :

0	3	7	5
2	0	6	4
3	1	0	5
5	3	2	0

25. Write a program to construct a minimal spanning tree of a graph

Code :

```
#include <bits/stdc++.h>
using namespace std;

class DSU
{
    int *parent;
    int *rank;

public:
    DSU(int n)
    {
        parent = new int[n];
        rank = new int[n];

        for (int i = 0; i < n; i++)
        {
            parent[i] = -1;
            rank[i] = 1;
        }
    }

    // Find function
    int find(int i)
    {
        if (parent[i] == -1)
            return i;

        return parent[i] = find(parent[i]);
    }

    // union function
    void unite(int x, int y)
    {
```

```

int s1 = find(x);
int s2 = find(y);

if (s1 != s2)
{
    if (rank[s1] < rank[s2])
    {
        parent[s1] = s2;
        rank[s2] += rank[s1];
    }
    else
    {
        parent[s2] = s1;
        rank[s1] += rank[s2];
    }
}
};

class Graph
{
    vector<vector<int>> edgelist;
    int V;

public:
    Graph(int V) { this->V = V; }

    void addEdge(int x, int y, int w)
    {
        edgelist.push_back({w, x, y});
    }

    void kruskals_mst()
    {
        // 1. Sort all edges

```

```

sort(edgelist.begin(), edgelist.end());

// Initialize the DSU
DSU s(V);
int ans = 0;
cout << "Following are the edges in the "
    "constructed MST"
    << endl;
for (auto edge : edgelist)
{
    int w = edge[0];
    int x = edge[1];
    int y = edge[2];

    // take that edge in MST if it does form a cycle
    if (s.find(x) != s.find(y))
    {
        s.unite(x, y);
        ans += w;
        cout << x << " -- " << y << " == " << w
            << endl;
    }
}
cout << "Minimum Cost Spanning Tree: " << ans;
}

};

int main()
{

Graph g(4);
g.addEdge(0, 1, 10);
g.addEdge(1, 3, 15);
g.addEdge(2, 3, 4);
g.addEdge(2, 0, 6);
g.addEdge(0, 3, 5);

```

```
g.kruskals_mst();
```

```
return 0;
```

```
}
```

```
Following are the edges in the constructed MST
```

```
2 -- 3 == 4
```

```
0 -- 3 == 5
```

```
0 -- 1 == 10
```

```
Minimum Cost Spanning Tree: 19%
```