`PROGRAMS FOR DATA STRUCTURES LAB EXAM, VJCET`

```
// ***POLYNOMIAL ADDITION***

#include<stdio.h>
#include<math.h>

/*
  This structure is used to store a polynomial term. An array of such terms
represents a
  polynomial.
  The "coeff" element stores the coefficient of a term in the polynomial,while
  the "exp"   element stores the exponent.

*/
struct poly
{
  float coeff;
  int exp;
};


//declaration of polynomials
struct poly a[50],b[50],c[50],d[50];

int main()
{
 int i;
 int deg1,deg2;      //stores degrees of the polynomial
 int k=0,l=0,m=0;


 printf("Enter the highest degree of poly1:");
 scanf("%d",&deg1);

 //taking polynomial terms from the user
 for(i=0;i<=deg1;i++)
 {

    //entering values in coefficient of the polynomial terms
    printf("\nEnter the coeff of x^%d :",i);
    scanf("%f",&a[i].coeff);

    //entering values in exponent of the polynomial terms
    a[k++].exp = i;
 }



 //taking second polynomial from the user
 printf("\nEnter the highest degree of poly2:");
 scanf("%d",&deg2);

 for(i=0;i<=deg2;i++)
 {
       printf("\nEnter the coeff of x^%d :",i);
       scanf("%f",&b[i].coeff);

       b[l++].exp = i;
 }


 //printing first polynomial
  printf("\nExpression 1 = %.1f",a[0].coeff);
  for(i=1;i<=deg1;i++)
  {
    printf("+ %.1fx^%d",a[i].coeff,a[i].exp);
  }


  //printing second polynomial
  printf("\nExpression 2 = %.1f",b[0].coeff);
   for(i=1;i<=deg2;i++)
    {
      printf("+ %.1fx^%d",b[i].coeff,b[i].exp);
    }


//Adding the polynomials
 if(deg1>deg2)
    {
         for(i=0;i<=deg2;i++)
          {
            c[m].coeff = a[i].coeff + b[i].coeff;
            c[m].exp = a[i].exp;
            m++;
          }

          for(i=deg2+1;i<=deg1;i++)
          {
            c[m].coeff = a[i].coeff;
            c[m].exp = a[i].exp;
            m++;
          }

    }
 else
  {
    for(i=0;i<=deg1;i++)
     {
       c[m].coeff = a[i].coeff + b[i].coeff;
       c[m].exp = a[i].exp;
       m++;
     }

    for(i=deg1+1;i<=deg2;i++)
    {
      c[m].coeff = b[i].coeff;
      c[m].exp = b[i].exp;
      m++;
    }
  }
  //printing the sum of the two polynomials
  printf("\nExpression after additon  = %.1f",c[0].coeff);
  for(i=1;i<m;i++)
  {
     printf("+ %.1fx^%d",c[i].coeff,c[i].exp);
   }

  return 0;

}

// ***CIRCULAR QUEUE***

#include <stdio.h>
#define size 5

void insertq(int[], int);
void deleteq(int[]);
void display(int[]);

int front =  - 1;
int rear =  - 1;

int main()
{
    int n, ch;
    int queue[size];
    do
    {
        printf("\n\n Circular Queue:\n1. Insert \n2. Delete\n3. Display\n0.
Exit");
        printf("\nEnter Choice 0-3? : ");
        scanf("%d", &ch);
        switch (ch)
        {
            case 1:
                printf("\nEnter number: ");
                scanf("%d", &n);
                insertq(queue, n);
                break;
            case 2:
                deleteq(queue);
                break;
            case 3:
                display(queue);
                break;
        }
    }while (ch != 0);
}


void insertq(int queue[], int item)
{
    if ((front == 0 && rear == size - 1) || (front == rear + 1))
    {
        printf("queue is full");
        return;
    }
    else if (rear ==  - 1)
    {
        rear++;
        front++;
    }
    else if (rear == size - 1 && front > 0)
    {
        rear = 0;
    }
    else
    {
        rear++;
    }
    queue[rear] = item;
}

void display(int queue[])
{
    int i;
    printf("\n");
    if (front > rear)
    {
        for (i = front; i < size; i++)
        {
            printf("%d ", queue[i]);
        }
        for (i = 0; i <= rear; i++)
            printf("%d ", queue[i]);
    }
    else
    {
        for (i = front; i <= rear; i++)
            printf("%d ", queue[i]);
    }
}

void deleteq(int queue[])
{
    if (front ==  - 1)
    {
        printf("Queue is empty ");
    }
    else if (front == rear)
    {
        printf("\n %d deleted", queue[front]);
        front =  - 1;
        rear =  - 1;
    }
    else
    {
        printf("\n %d deleted", queue[front]);
        front++;
    }
}

// ***DOUBLE ENDED QUEUE***
```

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

int deque[MAX];
int left = -1, right = -1;

void insert_right(void);
void insert_left(void);
void delete_right(void);
void delete_left(void);
void display(void);

int main()
{
    int choice;
    do
    {
        printf("\n1.Insert at right ");
        printf("\n2.Insert at left ");
        printf("\n3.Delete from right ");
        printf("\n4.Delete from left ");
        printf("\n5.Display ");
        printf("\n6.Exit");
        printf("\n\nEnter your choice ");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            insert_right();
            break;
        case 2:
            insert_left();
            break;
        case 3:
            delete_right();
            break;
        case 4:
            delete_left();
            break;
        case 5:
            display();
            break;
        }
    } while (choice != 6);
    getch();
    return 0;
}
void insert_right()
{
    int val;
    printf("\nEnter the value to be added ");
    scanf("%d", &val);
    if ((left == 0 && right == MAX - 1) || (left == right + 1))
    {
        printf("\nOVERFLOW");
    }
    if (left == -1)
    {
        left = 0;
        right = 0;
    }
    else
    {
        if (right == MAX - 1)
            right = 0;
        else
            right = right + 1;
    }
    deque[right] = val;
}
void insert_left()
{
    int val;
    printf("\nEnter the value to be added ");
    scanf("%d", &val);
    if ((left == 0 && right == MAX - 1) || (left == right + 1))
    {
        printf("\nOVERFLOW");
    }
    if (left == -1)
    {
        left = 0;
        right = 0;
    }
    else
    {
        if (left == 0)
            left = MAX - 1;
        else
            left = left - 1;
    }
    deque[left] = val;
}
void delete_right()
{
    if (left == -1)
    {
        printf("\nUNDERFLOW");
        return;
    }
    printf("\nThe deleted element is %d\n", deque[right]);
    if (left == right)
    {
        left = -1;
        right = -1;
    }
    else
    {
        if (right == 0)
            right = MAX - 1;
        else
            right = right - 1;
    }
}
```

```c
void delete_left()
{
    if (left == -1)
    {
        printf("\nUNDERFLOW");
        return;
    }
    printf("\nThe deleted element is %d\n", deque[left]);
    if (left == right)
    {
        left = -1;
        right = -1;
    }
    else
    {
        if (left == MAX - 1)
            left = 0;
        else
            left = left + 1;
    }
}
void display()
{
    int front = left, rear = right;
    if (front == -1)
    {
        printf("\nQueue is Empty\n");
        return;
    }
    printf("\nThe elements in the queue are: ");
    if (front <= rear)
    {
        while (front <= rear)
        {
            printf("%d\t", deque[front]);
            front++;
        }
    }
    else
    {
        while (front <= MAX - 1)
        {
            printf("%d\t", deque[front]);
            front++;
        }
        front = 0;
        while (front <= rear)
        {
            printf("%d\t", deque[front]);
            front++;
        }
    }
    printf("\n");
}

// ***LINKED LIST***

#include <stdio.h>
```

```c
#include <stdlib.h>
struct node
{
    int data;
    struct node *next;
};
struct node *head;

void beginsert();
void lastinsert();
void randominsert();
void begin_delete();
void last_delete();
void random_delete();
void display();
void search();
int main()
{
    int choice = 0;
    while (choice != 9)
    {
        printf("\n\n*********MainMenu*********\n");
        printf("\nChooseoneoptionfromthefollowinglist...\n");
        printf("\n===============================================\n");

printf("\n1.Insertinbegining\n2.Insertatlast\n3.Insertatanyrandomlocation\n4.Del
etefromBeginning\n5.Deletefromlast\n6.Deletenodeafterspecifiedlocation\n7.Search
foranelement\n8.Show\n9.Exit\n");
        printf("\nEnteryourchoice?\n");
        scanf("\n%d", &choice);
        switch (choice)
        {
        case 1:
            beginsert();
            break;
        case 2:
            lastinsert();
            break;
        case 3:
            randominsert();
            break;
        case 4:
            begin_delete();
            break;
        case 5:
            last_delete();
            break;
        case 6:
            random_delete();
            break;
        case 7:
            search();
            break;
        case 8:
            display();
            break;
        case 9:
            exit(0);
            break;
        default:
            printf("Pleaseentervalidchoice..");
        }
    }
}
void beginsert()
{
    struct node *ptr;
    int item;
    ptr = (struct node *)malloc(sizeof(struct node *));
    if (ptr == NULL)
    {
        printf("\nOVERFLOW");
    }
    else
    {
        printf("\nEntervalue\n");
        scanf("%d", &item);
        ptr->data = item;
        ptr->next = head;
        head = ptr;
        printf("\nNodeinserted");
    }
}
void lastinsert()
{
    struct node *ptr, *temp;
    int item;
    ptr = (struct node *)malloc(sizeof(struct node));
    if (ptr == NULL)
    {
        printf("\nOVERFLOW");
    }
    else
    {
        printf("\nEntervalue?\n");
        scanf("%d", &item);
        ptr->data = item;
        if (head == NULL)
        {
            ptr->next = NULL;
            head = ptr;
            printf("\nNodeinserted");
        }
        else
        {
            temp = head;
            while (temp->next != NULL)
            {
                temp = temp->next;
            }
            temp->next = ptr;
            ptr->next = NULL;
            printf("\nNodeinserted");
        }
    }
}
```

```
void randominsert()
{
    int i, loc, item;
    struct node *ptr, *temp;
    ptr = (struct node *)malloc(sizeof(struct node));
    if (ptr == NULL)
    {
        printf("\nOVERFLOW");
    }
    else
    {
        printf("\nEnterelementvalue");
        scanf("%d", &item);
        ptr->data = item;
        printf("\nEnterthelocationafterwhichyouwanttoinsert");
        scanf("\n%d", &loc);
        temp = head;
        for (i = 0; i < loc; i++)
        {
            temp = temp->next;
            if (temp == NULL)
            {
                printf("\ncan'tinsert\n");
                return;
            }
        }
        ptr->next = temp->next;
        temp->next = ptr;
        printf("\nNodeinserted");
    }
}
void begin_delete()
{
    struct node *ptr;
    if (head == NULL)
    {
        printf("\nListisempty\n");
    }
    else
    {
        ptr = head;
        head = ptr->next;
        free(ptr);
        printf("\nNodedeletedfromthebegining...\n");
    }
}
void last_delete()
{
    struct node *ptr, *ptr1;
    if (head == NULL)
    {
        printf("\nlistisempty");
    }
    else if (head->next == NULL)
    {
        head = NULL;
        free(head);
        printf("\nOnlynodeofthelistdeleted...\n");
```

```c
    }
    else
    {
        ptr = head;
        while (ptr->next != NULL)
        {
            ptr1 = ptr;
            ptr = ptr->next;
        }
        ptr1->next = NULL;
        free(ptr);
        printf("\nDeletedNodefromthelast...\n");
    }
}
void random_delete()
{
    struct node *ptr, *ptr1;
    int loc, i;
    printf("\nEnterthelocationofthenodeafterwhichyouwanttoperformdeletion\n");
    scanf("%d", &loc);
    ptr = head;
    for (i = 0; i < loc; i++)
    {
        ptr1 = ptr;
        ptr = ptr->next;
        if (ptr == NULL)
        {
            printf("\nCan'tdelete");
            return;
        }
    }
    ptr1->next = ptr->next;
    free(ptr);
    printf("\nDeletednode%d", loc + 1);
}
void search()
{
    struct node *ptr;
    int item, i = 0, flag;
    ptr = head;
    if (ptr == NULL)
    {
        printf("\nEmptyList\n");
    }
    else
    {
        printf("\nEnteritemwhichyouwanttosearch?\n");
        scanf("%d", &item);
        while (ptr != NULL)
        {
            if (ptr->data == item)
            {
                printf("itemfoundatlocation%d", i + 1);
                flag = 0;
            }
            else
            {
                flag = 1;
            }
            i++;
            ptr = ptr->next;
        }
        if (flag == 1)
        {
            printf("Itemnotfound\n");
        }
    }
}
void display()
{
    struct node *ptr;
    ptr = head;
    if (ptr == NULL)
    {
        printf("Nothingtoprint");
    }
    else
    {
        printf("\nprintingvalues.....\n");
        while (ptr != NULL)
        {
            printf("\n%d", ptr->data);
            ptr = ptr->next;
        }
    }
}

// ***STACK USING LINKED LIST***

#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 display();
void create();

int count = 0;

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

    printf("\n 1 - Push");
    printf("\n 2 - Pop");
    printf("\n 3 - Display");
    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:
            display();
            break;
        default:
            printf(" Wrong choice, Please enter correct choice  ");
            break;
        }
    }
}
void create()
{
    top = NULL;
}
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;
```

```c
    }
}
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--;
}

// ***QUEUE USING LINKED LIST***

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

struct Node
{
    int data;
    struct Node *next;
} *front = NULL, *rear = NULL;

void insert(int);
void dlt();
void display();

int main()
{
    int choice, value;
    printf("\n:: Queue Implementation using Linked List ::\n");
    while (1)
    {
        printf("\n****** MENU ******\n");
        printf("1. Insert\n2. Delete\n3. Display\n4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            printf("Enter the value to be insert: ");
            scanf("%d", &value);
            insert(value);
            break;
        case 2:
            dlt();
            break;
        case 3:
            display();
            break;
        case 4:
            exit(0);
        default:
            printf("\nWrong selection!!! Please try again!!!\n");
        }
    }
}
void insert(int value)
{
    struct Node *newNode;
    newNode = (struct Node *)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = NULL;
    if (front == NULL)
        front = rear = newNode;
    else
    {
        rear->next = newNode;
        rear = newNode;
    }
    printf("\nInsertion is Success!!!\n");
}
void dlt()
{
    if (front == NULL)
        printf("\nQueue is Empty!!!\n");
    else
    {
        struct Node *temp = front;
        front = front->next;
        printf("\nDeleted element: %d\n", temp->data);
        free(temp);
    }
}
void display()
{
    if (front == NULL)
        printf("\nQueue is Empty!!!\n");
    else
    {
        struct Node *temp = front;
        while (temp->next != NULL)
        {
            printf("%d--->", temp->data);
            temp = temp->next;
        }
        printf("%d--->NULL\n", temp->data);
    }
}

// ***REVERSE A QUEUE USING STACK***

#include <stdio.h>
#include <conio.h>
int f = -1, r = -1;
int q[50];
void enqueue(int data, int l)
{
```

```c
    if (r == l - 1)
    {
        printf("Queue is full");
    }
    else if ((f == -1) && (r == -1))
    {
        f = r = 0;
        q[r] = data;
    }
    else
    {
        r++;
        q[r] = data;
    }
}
void print()
{
    int i;
    for (i = f; i <= r; i++)
    {
        printf("\n%d", q[i]);
    }
}
void reverse()
{
    int i, j, t;
    for (i = f, j = r; i < j; i++, j--)
    {
        t = q[i];
        q[i] = q[j];
        q[j] = t;
    }
}
int main()
{
    int n, i = 0, t;
    printf("Enter the size of Queue");
    scanf("%d", &n);
    printf("\nEnter the data for Queue");
    while (i < n)
    {
        scanf("%d", &t);
        enqueue(t, n);
        i++;
    }
    printf("\nQueue which you have entered:-");
    print();
    reverse();
    printf("\nQueue after reversing:-");
    print();
}

// ***CHECK PALINDROME FOR STRING,USING DOUBLY LINKED LIST***

#include <stdio.h>
#include <stdlib.h>
struct Node
{
    int data;
    struct Node* next;
};

void push(struct Node** head, int data)
{
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));

    newNode->data = data;
    newNode->next = *head;
    *head = newNode;
}
int checkPalindrome(struct Node** left, struct Node* right)
{

    if (right == NULL) {
        return 1;
    }

    int res = checkPalindrome(left, right->next) &&
            ((*left)->data == right->data);
    (*left) = (*left)->next;

    return res;
}
int checkPalin(struct Node* head)
{
    return checkPalindrome(&head, head);
}

int main(void)
{
    int keys[] = { 1, 3, 5, 3, 1};
    int n = sizeof(keys) / sizeof(keys[0]);

    struct Node* head = NULL;
    for (int i = n - 1; i >= 0; i--) {
        push(&head, keys[i]);
    }

    if (checkPalin(head)) {
        printf("The linked list is a palindrome");
    } else {
        printf("The linked list is not a palindrome");
    }

    return 0;
}

// ***LINEAR SEARCH***

#include <stdio.h>

int main()
{
    int a[20], i, x, n;
    printf("How many elements?");
    scanf("%d", &n);
```

```c
    printf("Enter array elements:\n");
    for (i = 0; i < n; ++i)
        scanf("%d", &a[i]);

    printf("\nEnter element to search:");
    scanf("%d", &x);

    for (i = 0; i < n; ++i)
        if (a[i] == x)
            break;

    if (i < n)
        printf("Element found at index %d", i);
    else
        printf("Element not found");

    return 0;
}

// ***BINARY SEARCH***

#include <stdio.h>

int main()
{
    int arr[50], i, n, x, flag = 0, first, last, mid;

    printf("Enter size of array:");
    scanf("%d", &n);
    printf("\nEnter array element(ascending order)\n");

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

    printf("\nEnter the element to search:");
    scanf("%d", &x);

    first = 0;
    last = n - 1;

    while (first <= last)
    {
        mid = (first + last) / 2;

        if (x == arr[mid])
        {
            flag = 1;
            break;
        }
        else if (x > arr[mid])
            first = mid + 1;
        else
            last = mid - 1;
    }

    if (flag == 1)
        printf("\nElement found at position %d", mid + 1);
    else
        printf("\nElement not found");

    return 0;
}

// ***MERGE SORT***

#include <stdio.h>

void mergesort(int a[], int i, int j);
void merge(int a[], int i1, int j1, int i2, int j2);

int main()
{
    int a[30], n, i;
    printf("Enter no of elements:");
    scanf("%d", &n);
    printf("Enter array elements:");

    for (i = 0; i < n; i++)
        scanf("%d", &a[i]);

    mergesort(a, 0, n - 1);

    printf("\nSorted array is :");
    for (i = 0; i < n; i++)
        printf("%d ", a[i]);

    return 0;
}

void mergesort(int a[], int i, int j)
{
    int mid;

    if (i < j)
    {
        mid = (i + j) / 2;
        mergesort(a, i, mid);         //left recursion
        mergesort(a, mid + 1, j);     //right recursion
        merge(a, i, mid, mid + 1, j); //merging of two sorted sub-arrays
    }
}

void merge(int a[], int i1, int j1, int i2, int j2)
{
    int temp[50]; //array used for merging
    int i, j, k;
    i = i1; //beginning of the first list
    j = i2; //beginning of the second list
    k = 0;

    while (i <= j1 && j <= j2) //while elements in both lists
    {
        if (a[i] < a[j])
            temp[k++] = a[i++];
        else
```

```
            temp[k++] = a[j++];
    }

    while (i <= j1) //copy remaining elements of the first list
        temp[k++] = a[i++];

    while (j <= j2) //copy remaining elements of the second list
        temp[k++] = a[j++];

    //Transfer elements from temp[] back to a[]
    for (i = i1, j = 0; i <= j2; i++, j++)
        a[i] = temp[j];
}

// ***QUICK SORT***

#include <stdio.h>

void quick_sort(int[], int, int);
int partition(int[], int, int);

int main()
{
    int a[50], n, i;
    printf("How many elements?");
    scanf("%d", &n);
    printf("\nEnter array elements:");

    for (i = 0; i < n; i++)
        scanf("%d", &a[i]);

    quick_sort(a, 0, n - 1);
    printf("\nArray after sorting:");

    for (i = 0; i < n; i++)
        printf("%d ", a[i]);

    return 0;
}

void quick_sort(int a[], int l, int u)
{
    int j;
    if (l < u)
    {
        j = partition(a, l, u);
        quick_sort(a, l, j - 1);
        quick_sort(a, j + 1, u);
    }
}

int partition(int a[], int l, int u)
{
    int v, i, j, temp;
    v = a[l];
    i = l;
    j = u + 1;
```

```c
    do
    {
        do
            i++;

        while (a[i] < v && i <= u);

        do
            j--;
        while (v < a[j]);

        if (i < j)
        {
            temp = a[i];
            a[i] = a[j];
            a[j] = temp;
        }
    } while (i < j);

    a[l] = a[j];
    a[j] = v;

    return (j);
}

// ***BREADTH FIRST SEARCH***

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

#define MAX 100

#define initial 1
#define waiting 2
#define visited 3

int n;
int adj[MAX][MAX];
int state[MAX];
void create_graph();
void BF_Traversal();
void BFS(int v);

int queue[MAX], front = -1, rear = -1;
void insert_queue(int vertex);
int delete_queue();
int isEmpty_queue();

int main()
{
    create_graph();
    BF_Traversal();
    return 0;
}

void BF_Traversal()
{
    int v;
    for (v = 0; v < n; v++)
        state[v] = initial;

    printf("Enter Start Vertex for BFS: \n");
    scanf("%d", &v);
    BFS(v);
}

void BFS(int v)
{
    int i;

    insert_queue(v);
    state[v] = waiting;

    while (!isEmpty_queue())
    {
        v = delete_queue();
        printf("%d ", v);
        state[v] = visited;

        for (i = 0; i < n; i++)
        {
            if (adj[v][i] == 1 && state[i] == initial)
            {
                insert_queue(i);
                state[i] = waiting;
            }
        }
    }
    printf("\n");
}

void insert_queue(int vertex)
{
    if (rear == MAX - 1)
        printf("Queue Overflow\n");
    else
    {
        if (front == -1)
            front = 0;
        rear = rear + 1;
        queue[rear] = vertex;
    }
}

int isEmpty_queue()
{
    if (front == -1 || front > rear)
        return 1;
    else
        return 0;
}

int delete_queue()
{
    int delete_item;
    if (front == -1 || front > rear)
    {
        printf("Queue Underflow\n");
        exit(1);
    }

    delete_item = queue[front];
    front = front + 1;
    return delete_item;
}

void create_graph()
{
    int count, max_edge, origin, destin;

    printf("Enter number of vertices : ");
    scanf("%d", &n);
    max_edge = n * (n - 1);

    for (count = 1; count <= max_edge; count++)
    {
        printf("Enter edge %d( -1 -1 to quit ) : ", count);
        scanf("%d %d", &origin, &destin);

        if ((origin == -1) && (destin == -1))
            break;

        if (origin >= n || destin >= n || origin < 0 || destin < 0)
        {
            printf("Invalid edge!\n");
            count--;
        }
        else
        {
            adj[origin][destin] = 1;
        }
    }
}

// ***DEPTH FIRST SEARCH (ADJACENCY MATRIX)***

#include <stdio.h>

void DFS(int);
int G[10][10], visited[10], n; //n is no of vertices and graph is sorted in
array G[10][10]

void main()
{
    int i, j;
    printf("Enter number of vertices:");

    scanf("%d", &n);

    //read the adjecency matrix
    printf("\nEnter adjecency matrix of the graph:");

    for (i = 0; i < n; i++)
```

```c
        for (j = 0; j < n; j++)
            scanf("%d", &G[i][j]);

    //visited is initialized to zero
    for (i = 0; i < n; i++)
        visited[i] = 0;

    DFS(0);
}

void DFS(int i)
{
    int j;
    printf("\n%d", i);
    visited[i] = 1;

    for (j = 0; j < n; j++)
        if (!visited[j] && G[i][j] == 1)
            DFS(j);
}

// ***DEPTH FIRST SEARCH (ADJACENCY LIST)***

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

typedef struct node
{
    struct node *next;
    int vertex;
} node;

node *G[20];
//heads of linked list
int visited[20];
int n;
void read_graph();
//create adjacency list
void insert(int, int);
//insert an edge (vi,vj) in te adjacency list
void DFS(int);

void main()
{
    int i;
    read_graph();
    //initialised visited to 0

    for (i = 0; i < n; i++)
        visited[i] = 0;

    DFS(0);
}

void DFS(int i)
{
    node *p;
    printf("\n%d", i);
    p = G[i];
    visited[i] = 1;
    while (p != NULL)
    {
        i = p->vertex;

        if (!visited[i])
            DFS(i);
        p = p->next;
    }
}

void read_graph()
{
    int i, vi, vj, no_of_edges;
    printf("Enter number of vertices:");

    scanf("%d", &n);

    //initialise G[] with a null

    for (i = 0; i < n; i++)
    {
        G[i] = NULL;
        //read edges and insert them in G[]

        printf("Enter number of edges:");
        scanf("%d", &no_of_edges);

        for (i = 0; i < no_of_edges; i++)
        {
            printf("Enter an edge(u,v):");
            scanf("%d%d", &vi, &vj);
            insert(vi, vj);
        }
    }
}

void insert(int vi, int vj)
{
    node *p, *q;

    //acquire memory for the new node
    q = (node *)malloc(sizeof(node));
    q->vertex = vj;
    q->next = NULL;

    //insert the node in the linked list number vi
    if (G[vi] == NULL)
        G[vi] = q;
    else
    {
        //go to end of the linked list
        p = G[vi];

        while (p->next != NULL)
            p = p->next;
        p->next = q;
    }
}

// ***BINARY SEARCH TREES***

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

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

node *create();
void insert(node *, node *);
void preorder(node *);

int main()
{
    char ch;
    node *root = NULL, *temp;

    do
    {
        temp = create();
        if (root == NULL)
            root = temp;
        else
            insert(root, temp);

        printf("nDo you want to enter more(y/n)?");
        getchar();
        scanf("%c", &ch);
    } while (ch == 'y' | ch == 'Y');

    printf("nPreorder Traversal: ");
    preorder(root);
    return 0;
}

node *create()
{
    node *temp;
    printf("nEnter data:");
    temp = (node *)malloc(sizeof(node));
    scanf("%d", &temp->data);
    temp->left = temp->right = NULL;
    return temp;
}

void insert(node *root, node *temp)
{
    if (temp->data < root->data)
    {
        if (root->left != NULL)
```

```
            insert(root->left, temp);
        else
            root->left = temp;
    }

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

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