```c
//Implement mergesort
#include <stdio.h>
#define MAX 100

void merge(int A[], int start, int mid, int end)
{
    int temp[MAX];
    for(int i=start ; i<=end ; i++)
    {
        temp[i] = A[i];
    }
    int k=start,i,j;
    for(i=start,j=mid+1 ; i<=mid && j<=end ; )
    {
        if(temp[i] <= temp[j])
        {
            A[k++] = temp[i++];
        }
        else
        {
            A[k++] = temp[j++];
        }
    }
    while(i <= mid)
        A[k++] = temp[i++];
    while(j <= end)
        A[k++] = temp[j++];
}

void mergesort(int A[], int start, int end)
{
    if(start < end)
    {
        int mid = (start+end)/2;;
        mergesort(A, start, mid);
        mergesort(A, mid+1, end);
        merge(A, start, mid, end);
    }
}

int main()
{
    int a[] = {2,23,4,15,62,13,7,8};
    printf("Original array\n");
    for(int i=0 ; i<8 ; i++)
        printf("%6d", a[i]);
    printf("\n");
    mergesort(a, 0, 7);
    printf("Sorted array\n");
    for(int i=0 ; i<8 ; i++)
        printf("%6d", a[i]);
    printf("\n");
}
/*OUTPUT
Original array
     2    23     4    15    62    13     7     8
Sorted array
     2     4     7     8    13    15    23    62
*/
```

```c
//Implement quicksort
#include <stdio.h>
#define MAX 100

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

int get_partion(int A[], int start , int end)
{
    int pivot = A[end];
    int i=start-1;
    for(int j=start ; j<end ; j++)
    {
        if(A[j] <= pivot)
        {
            i++;
            swap(A[i], A[j]);
        }
    }
    swap(A[end], A[i+1]);
    return i+1;
}

void quicksort(int A[], int start, int end)
{
    if(start < end)
    {
        int partion = get_partion(A, start, end);
        quicksort(A, start, partion-1);
        quicksort(A, partion+1, end);
    }
}

int main()
{
    int a[] = {2,23,4,15,62,13,7,8};
    printf("Original array\n");
    for(int i=0 ; i<8 ; i++)
        printf("%6d", a[i]);
    printf("\n");
    quicksort(a, 0, 7);
    printf("Sorted array\n");
    for(int i=0 ; i<8 ; i++)
        printf("%6d", a[i]);
    printf("\n");
}
/*
OUTPUT
Original array
     2    23     4    15    62    13     7     8
Sorted array
     2     4     7     8    13    15    23    62
*/
```

```c
//Implement HeapSort
#include <stdio.h>

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

void max_heapify(int A[], int index, int heap_size)
{
    int left = 2*index+1, right = 2*index+2, largest=index;
    if (left <= heap_size && A[left] > A[index])
    {
        largest = left;
    }
    if (right <= heap_size && A[right] > A[index])
    {
        largest = right;
    }
    if(left <= heap_size && right <= heap_size)
    {
        if(A[left] > A[right] && A[left] > A[index])
            largest = left;
        else if(A[right] > A[left] && A[right] > A[index])
            largest = right;
    }
    if (largest != index)
    {
        swap(A[index], A[largest]);
        max_heapify(A, largest, heap_size);
    }
}

void build_maxheap(int A[], int heap_size)
{
    for(int i=heap_size/2 +1 ; i>=0 ; i--)
        max_heapify(A,i,heap_size);
}

void heapsort(int A[], int heap_size)
{
    build_maxheap(A, heap_size);
    for(int i=heap_size ; i>=1 ; i--)
    {
        swap(A[0], A[i]);
        heap_size--;
        max_heapify(A, 0, heap_size);
    }
}

int main()
{
    int a[10] = {4,1,3,2,16,9,10,14,8,7};
    printf("Original array\n");
    for(int i=0 ; i<10 ; i++)
    {
        printf("%6d", a[i]);
    }
    heapsort(a, 9);
    printf("\n");
    printf("Sorted Array\n");
    for(int i=0 ; i<10 ; i++)
    {
        printf("%6d", a[i]);
    }
    printf("\n");
}
```

```
/*
OUTPUT
Original array
     4     1     3     2    16     9    10    14     8     7
Sorted Array
     1     2     3     4     7     8     9    10    14    16
*/
```

```c
//Implement a stack using an array.
#include <stdio.h>
#define MAX 50
#define INF -100000

struct stack
{
    int array[50];
    int top;
    int size;
};

void push(struct stack &s)
{
    if(s.top != s.size)
    {
        int temp;
        printf("Enter the input : ");
        scanf("%d", &temp);
        s.array[++s.top] = temp;
    }
    else
    {
        printf("Stack Overflow\n");
    }
}

int pop(struct stack &s)
{
    if(s.top != -1)
    {
        int temp = s.array[s.top--];
        return temp;
    }
    else
    {
        printf("Stack Underflow\n");
        return INF;
    }
}

void display(struct stack &s)
{
    if(s.top != -1)
    {
        for(int i=s.top ; i>=0 ; i--)
        {
            printf("%d->", s.array[i]);
        }
        printf("\n");
    }
}

int main()
{
    struct stack s;
    s.top = -1;
    s.size = MAX;
    push(s);
    push(s);
    push(s);
    push(s);
    display(s);
    printf("After removing 1 element from stack\n");
    int c = pop(s);
    display(s);
}

/*OUTPUT
```

```
Enter the input : 1
Enter the input : 2
Enter the input : 3
Enter the input : 4
4->3->2->1->
After removing 1 element from stack
3->2->1->
*/
```

```c
//Implement stack using linked list
#include <stdio.h>
#include <stdlib.h>

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

node *get_new_node()
{
    node *p;
    p = (node *)malloc(sizeof(node));
    return p;
}

void push(node **top, int x_val)
{
    node *p = get_new_node();
    p->x = x_val;
    p->next = *top;
    *top = p;
}

int pop(node **top)
{
    if(*top == NULL)
    {
        printf("Stack Underflow\n");
        return -1;
    }
    node *p = *top;
    *top = (*top)->next;
    return p->x;
}

void display(node *top)
{
    node *p = top;
    while(p != NULL)
    {
        printf("%d -> ", p->x);
        p = p->next;
    }
    printf("\n");
}

int main()
{
    node *top;
    top = NULL;
    push(&top, 1);
    push(&top, 2);
    push(&top, 3);
    push(&top, 4);
    display(top);
    printf("After popping once\n");
    pop(&top);
    display(top);
}
/*
OUTPUT
4 -> 3 -> 2 -> 1 ->
After popping once
3 -> 2 -> 1 ->
*/
```

```c
//Implent a circular array queue
#include <stdio.h>
#define MAX 5

struct caq
{
    int array[MAX];
    int front;
    int rear;
};

void push(caq *cir, int x)
{
    if((cir->rear + 1)%MAX == cir->front)
        return;
    else
    {
        if(cir->rear == MAX-1)
        {
            cir->rear = (cir->rear+1)%MAX;
            cir->array[cir->rear] = x;
            return;
        }
        cir->array[cir->rear] = x;
        cir->rear = (cir->rear+1)%MAX;
    }
}

int pop(caq *cir)
{
    if(cir->rear == cir->front)
        return -1;
    int rr = cir->array[cir->front];
    cir->front = (cir->front+1)%MAX;
    return rr;
}

void display(caq *cir)
{
    if(cir->rear == cir->front)
        return;
    if(cir->front < cir->rear)
    {
        for(int i=cir->front; i<cir->rear ; i++)
        {
            printf("%6d", cir->array[i]);
        }
    }
    else
    {
        for(int i=cir->front ; i<MAX-1 ; i++)
        {
            printf("%6d", cir->array[i]);
        }
        for(int i=0 ; i<=cir->rear ; i++)
        {
            printf("%6d", cir->array[i]);
        }
    }
    printf("\n");
}

int main(int argc, char const *argv[])
{
    caq cir;
    cir.front = cir.rear = 0;
    push(&cir, 1);
    push(&cir, 2);
    push(&cir, 3);
```

```
    push(&cir, 4);
    display(&cir);
    printf("After removing 1 element from queue\n");
    pop(&cir);
    display(&cir);
    push(&cir, 6);
    printf("After adding 1 element to queue\n");
    display(&cir);
}
/*OUTPUT
     1     2     3     4
After removing 1 element from queue
     2     3     4
After adding 1 element to queue
     2     3     4     6
*/
```

```
//Implement a priority queue using Heap
#include <stdio.h>
#define MAX 100
#define MIN -10000

struct Heap
{
    int A[MAX];
    int heap_size;
};

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

void max_heapify(Heap *heap, int index)
{
    int left = 2*index+1, right = 2*index+2, largest=index;
    if (left <= heap->heap_size && heap->A[left] > heap->A[index])
    {
        largest = left;
    }
    if (right <= heap->heap_size && heap->A[right] > heap->A[index])
    {
        largest = right;
    }
    if(left <= heap->heap_size && right <= heap->heap_size)
    {
        if(heap->A[left] > heap->A[right] && heap->A[left] > heap->A[index])
            largest = left;
        else if(heap->A[right] > heap->A[left] && heap->A[right] > heap->A[index])
            largest = right;
    }
    if (largest != index)
    {
        swap(heap->A[index], heap->A[largest]);
        max_heapify(heap, largest);
    }
}

void build_maxheap(Heap *heap)
{
    for(int i=heap->heap_size/2 +1 ; i>=0 ; i--)
        max_heapify(heap ,i);
}

int heap_maximum(Heap heap)
{
    return heap.A[0];
}

int extract_maximum(Heap *heap)
{
    if(heap->heap_size < 1)
    {
        printf("Heap Underflow\n");
        return -1;
    }
    int maxa = heap->A[0];
    heap->A[0] = heap->A[heap->heap_size];
    heap->heap_size--;
    max_heapify(heap, 0);
    return maxa;
}

void increase_key(Heap *heap, int index, int key)
```

```
{
    if (key < heap->A[index])
    {
        printf("Error\n");
        return;
    }
    heap->A[index] = key;
    while(index > 1 && heap->A[(index-1)/2] < heap->A[index])
    {
        swap(heap->A[index], heap->A[(index-1)/2]);
        index = (index-1)/2;
    }
}

void max_heap_insert(Heap *heap, int key)
{
    heap->heap_size++;
    heap->A[heap->heap_size] = MIN;
    increase_key(heap, heap->heap_size, key);
}

int main()
{
    Heap heap;
    for(int i=0 ; i<5 ; i++)
        heap.A[i] = i+1;
    heap.heap_size = 4;
    build_maxheap(&heap);
    printf("Maximum element in heap : %d\n", heap_maximum(heap));
    printf("After inserting 6 in heap\n");
    max_heap_insert(&heap, 6);
    printf("Maximum element in heap : %d\n",heap_maximum(heap));
}
/*
OUTPUT
Maximum element in heap : 5
After inserting 6 in heap
Maximum element in heap : 6
*/
```

```c
//Implement a Binary Search Tree(create, insert, delete, number of nodes, height,
traversal, copy)
#include <stdio.h>
#include <stdlib.h>

struct node
{
    int key;
    node *left, *right, *parent;
};

node *get_new_node()
{
    node *p;
    p = (node *)malloc(sizeof(node));
    return p;
}

int max(int a,int b)
{
    if (a>b)
        return a;
    else
        return b;
}

void tree_insert(node **bst, int val)
{
    node *p = *bst, *q;
    while(p != NULL)
    {
        q = p;
        if(p->key < val)
            p = p->right;
        else
            p = p->left;
    }
    p = get_new_node();
    p->key = val;
    p->left = NULL;
    p->right = NULL;
    p->parent = q;
    if (q == NULL)
    {
        *bst = p;
    }
    else
    {
        if(q->key < val)
            q->right = p;
        else
            q->left = p;
    }
}

node * tree_minimum(node *bst)
{
    node *p = bst;
    while(p->left !=NULL)
    {
        p = p->left;
    }
    return p;
}

node * tree_successor(node *x)
{
    if (x->right != NULL)
        return tree_minimum(x->right);
```

```
    node *y=x->parent;
    while(y!= NULL && x == y->right)
    {
        x = y;
        y = y->parent;
    }
    return y;
}

node * search(node *bst, int key)
{
    node *p = bst;
    while(p!=NULL)
    {
        if(p->key > key)
            p = p->left;
        else if(p->key < key)
            p = p->right;
        else
            return p;
    }
}

node *copy(node *bst)
{
    if(bst == NULL)
        return NULL;
    node *bst_copy = get_new_node();
    bst_copy->key = bst->key;
    bst_copy->left = copy(bst->left);
    bst_copy->right = copy(bst->right);
    return bst_copy;
}

void tree_delete(node **bst, node *z)
{
    node *y;
    if (z->left == NULL || z->right == NULL)
    {
        if(z ->left == NULL && z->right == NULL)
            y = NULL;
        else if(z->left != NULL)
            y = z->left;
        else if(z->right != NULL)
            y = z->right;
        if (z->parent->left == z)
        {
            z->parent->left = y;
        }
        else
        {
            z->parent->right = y;

        }
        free(z);
    }
    else
    {
        y = tree_successor(z);
        z->key = y->key;
        tree_delete(bst, y);
    }

}

void inorder(node *bst)
{
    if(bst != NULL)
    {
```

```
        inorder(bst->left);
        printf("%6d", bst->key);
        inorder(bst->right);
    }
}

void preorder(node *bst)
{
    if(bst != NULL)
    {
        printf("%6d", bst->key);
        preorder(bst->left);
        preorder(bst->right);
    }
}

int count_nodes(node *bst)
{
    if (bst == NULL)    return 0;
    return 1 + count_nodes(bst->left) + count_nodes(bst->right);
}

int height(node *bst)
{
    if(bst == NULL) return 0;
    return 1 + max(height(bst->left), height(bst->right));
}

int main()
{
    node *bst;
    bst = NULL;
    tree_insert(&bst, 5);
    tree_insert(&bst, 3);
    tree_insert(&bst, 2);
    tree_insert(&bst, 7);
    tree_insert(&bst, 8);
    printf("Original Preorder\n");
    preorder(bst);
    printf("\n");
    tree_delete(&bst, search(bst,5));
    printf("After Deletion Preorder\n");
    preorder(bst);
    printf("\n");
    printf("Total Nodes = %d\n", count_nodes(bst));
    printf("Height = %d\n", height(bst));

}

/*OUTPUT
Original Preorder
     5     3     2     7     8
After Deletion Preorder
     7     3     2     8
Total Nodes = 4
Height = 3
*/
```

```c
//Sparse Matrix operations
#include <stdio.h>
#include <string.h>

struct sparse
{
    int row,col;
    int value;
};

void matrixify(struct sparse spa_mat[])
{
    printf("Normal Matrix : \n");
    int mat[spa_mat[0].row][spa_mat[0].col];
    memset(mat, 0, sizeof(mat[0][0]) * spa_mat[0].row * spa_mat[0].col);
    for(int i=1 ; i<=spa_mat[0].value ; i++)
    {
        mat[spa_mat[i].row][spa_mat[i].col] = spa_mat[i].value;
    }
    for(int i=0 ; i<spa_mat[0].row ; i++)
    {
        for(int j=0 ; j<spa_mat[0].col ; j++)
        {
            printf("%6d", mat[i][j]);
        }
        printf("\n");
    }

}

void sparsify(int m,int n, int mat[10][10])
{
    printf("Sparse Matrix : \n");
    struct sparse spa_mat[50];
    spa_mat[0].row = m;
    spa_mat[0].col = n;
    int k=1;
    for(int i=0 ; i<m ; i++)
    {
        for(int j=0 ; j<n ; j++)
        {
            if(mat[i][j] != 0)
            {
                spa_mat[k].row = i;
                spa_mat[k].col = j;
                spa_mat[k].value = mat[i][j];
                k++;
            }
        }
    }
    spa_mat[0].value = k-1;
    for(int i=0 ; i<k ; i++)
    {
        printf("%d %6d %6d\n", spa_mat[i].row, spa_mat[i].col, spa_mat[i].value);
    }
    matrixify(spa_mat);
}

void input_spa_mat(struct sparse mat[])
{
    printf("m : ");
    scanf("%d", &mat[0].row);
    printf("n : ");
    scanf("%d", &mat[0].col);
    printf("k : ");
    scanf("%d", &mat[0].value);
    for(int i=1 ; i<= mat[0].value ; i++)
    {
        scanf("%d %d %d", &mat[i].row, &mat[i].col, &mat[i].value);
```

```
    }
}

void print_spa_mat(struct sparse mat[])
{
    for(int i=0 ; i<=mat[0].value ; i++)
    {
        printf("%d %6d %6d\n", mat[i].row, mat[i].col, mat[i].value);
    }
}

void add_sparse(struct sparse a[], struct sparse b[])
{
    int i,j,k;
    struct sparse c[50];
    c[0].value = 0;
    c[0].row = a[0].row;
    c[0].col = a[0].col;
    for(i=1,j=1,k=1 ; i<=a[0].value && j<=b[0].value ; )
    {
        if(a[i].row == b[j].row)
        {
            if(a[i].col == b[j].col)
            {
                c[k].row = a[i].row;
                c[k].col = a[i].col;
                c[k].value = a[i].value + b[j].value;
                c[0].value++;
                i++;
                j++;
                k++;
            }
            else if(a[i].col < b[j].col)
            {
                c[k].row = a[i].row;
                c[k].col = a[i].col;
                c[k].value = a[i].value;
                c[0].value++;
                i++;
                k++;
            }
            else
            {
                c[k].row = b[j].row;
                c[k].col = b[j].col;
                c[k].value = b[j].value;
                c[0].value++;
                j++;
                k++;
            }
        }
        else if(a[i].row < b[j].row)
        {
            c[k].row = a[i].row;
            c[k].col = a[i].col;
            c[k].value = a[i].value;
            c[0].value++;
            i++;
            k++;
        }
        else
        {
            c[k].row = b[j].row;
            c[k].col = b[j].col;
            c[k].value = b[j].value;
            c[0].value++;
            j++;
            k++;
        }
```

```
    }
    if(i<=a[0].value)
    {
        for(; i<=a[0].value ;)
        {
            c[k].row = a[i].row;
            c[k].col = a[i].col;
            c[k].value = a[i].value;
            c[0].value++;
            i++;
            k++;
        }
    }
    if(j<=b[0].value)
    {
        for(; i<=b[0].value ;)
        {
            c[k].row = b[j].row;
            c[k].col = b[j].col;
            c[k].value = b[j].value;
            c[0].value++;
            j++;
            k++;
        }
    }
    printf("\n");
    print_spa_mat(c);
}

void transpose_sparse(struct sparse s[], struct sparse s_trans[])
{
    s_trans[0].row = s[0].col;
    s_trans[0].col = s[0].row;
    s_trans[0].value = s[0].value;
    int k=1;
    if(s_trans[0].value > 0)
    {
        for (int i = 0; i < s[0].col; ++i)
        {
            for (int j = 1; j <= s[0].value; ++j)
            {
                if(s[j].col == i)
                {
                    s_trans[k].row = s[j].col;
                    s_trans[k].col = s[j].row;
                    s_trans[k].value = s[j].value;
                    k++;
                }
            }
        }
    }
}

int main()
{
   struct sparse a[50], b[50], c[50];
   printf("Input Matrix 1 \n");
   input_spa_mat(a);
   printf("Input Matrix 2 \n");
   input_spa_mat(b);
   printf("Matrix Addition\n");
   add_sparse(a,b);
   transpose_sparse(a,c);
   printf("Transpose of Matrix 1\n");
   print_spa_mat(c);
}
/*
OUTPUT
```

```
Input Matrix 1
m : 3
n : 3
k : 2
0 0 1
2 2 2
Input Matrix 2
m : 3
n : 3
k : 2
1 1 1
1 0 1
Matrix Addition
3      3      4
0      0      1
1      1      1
1      0      1
2      2      2
Transpose of Matrix 1
3      3      2
0      0      1
2      2      2
*/
```

```c
//Depth First Search of a Graph
#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);

int 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;
    }
}

/*
OUTPUT:
Enter number of vertices:8
Enter number of edges:10
Enter an edge(u,v):0 1
Enter an edge(u,v):0 2
Enter an edge(u,v):0 3
Enter an edge(u,v):0 4
Enter an edge(u,v):1 5
Enter an edge(u,v):2 5
Enter an edge(u,v):3 6
Enter an edge(u,v):4 6
Enter an edge(u,v):5 7
Enter an edge(u,v):6 7

0
1
5
7
2
3
6
4
*/
```

```c
//Implement Breadth First Search of a Graph
#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;
        }
    }
}
/*OUTPUT
Enter number of vertices : 9
Enter edge 1( -1 -1 to quit ) : 0 1
Enter edge 2( -1 -1 to quit ) : 0 3
Enter edge 3( -1 -1 to quit ) : 0 4
Enter edge 4( -1 -1 to quit ) : 1 2
Enter edge 5( -1 -1 to quit ) : 3 6
Enter edge 6( -1 -1 to quit ) : 4 7
Enter edge 7( -1 -1 to quit ) : 6 4
Enter edge 8( -1 -1 to quit ) : 6 7
```

```
Enter edge 9( -1 -1 to quit ) : 2 5
Enter edge 10( -1 -1 to quit ) : 4 5
Enter edge 11( -1 -1 to quit ) : 7 5
Enter edge 12( -1 -1 to quit ) : 7 8
Enter edge 13( -1 -1 to quit ) : -1 -1
Enter Start Vertex for BFS:
0
0 1 3 4 2 6 5 7 8
*/
```

```c
//Addtion & Multiplication of polynomials using stack
#include <stdio.h>
#include <stdlib.h>

struct term
{
    int coeff;
    int xpower;
    struct term *next;
};

struct polynomial_head
{
    struct term *start;
    int num_terms;
};

typedef struct term poly_term;
typedef struct polynomial_head poly_head;

poly_term *get_new_term()
{
    poly_term *t;
    t = (poly_term *)malloc(sizeof(poly_term));
    return t;
}

void term_insert(poly_head *h, int coeff, int xpower)
{
    poly_term *p,*q;
    p = get_new_term();
    p->coeff = coeff;
    p->xpower = xpower;
    if(h->start==NULL || (h->start != NULL && h->start->xpower < xpower))
    {
        p->next = h->start;
        h->start = p;
        h->num_terms++;
    }
    else if(h->start != NULL && h->start->xpower == xpower)
    {
        h->start->coeff += coeff;
    }
    else
    {
        q = h->start;
        while(q->next!=NULL && q->next->xpower > xpower)
            q = q->next;
        if(q->next!=NULL && q->next->xpower == xpower)
        {
            q->next->coeff += coeff;
        }
        else
        {
            p->next = q->next;
            q->next = p;
            h->num_terms++;
        }

    }

}

void displayPolynomial(poly_head h)
{
    poly_term *p = h.start;
    while(p!=NULL)
    {
        printf("(%d)x^%d", p->coeff, p->xpower);
```

```
        if(p->next!=NULL)
            printf(" + ");
        p = p->next;
    }
    printf("\n");
}

void new_poly(poly_head *h)
{
    int num_terms;
    printf("\nTerms in Polynomial : ");
    scanf("%d", &num_terms);
    printf("Enter The Polynomial : \n");
    int coeff, xpower;
    for (int i = 0; i < num_terms; ++i)
    {
        printf("Term %d : ", i+1);
        scanf("%d %d", &coeff, &xpower);
        term_insert(h, coeff, xpower);
    }
}

poly_head add(poly_head p1, poly_head p2)
{
    poly_head p3;
    p3.start = NULL;
    p3.num_terms = 0;
    poly_term *a = p1.start, *b = p2.start;
    while(a!=NULL && b!=NULL)
    {
        if(a->xpower > b->xpower)
        {
            term_insert(&p3, a->coeff, a->xpower);
            a = a->next;
        }
        else if(a->xpower < b->xpower)
        {
            term_insert(&p3, b->coeff, b->xpower);
            b = b->next;
        }
        else
        {
            term_insert(&p3, a->coeff+b->coeff, a->xpower);
            a = a->next;
            b = b->next;
        }
    }
    if(a!=NULL)
    {
        while(a!=NULL)
        {
            term_insert(&p3, a->coeff, a->xpower);
            a = a->next;
        }
    }
    else if(b!=NULL)
    {
        while(b!=NULL)
        {
            term_insert(&p3, b->coeff, b->xpower);
            b = b->next;
        }
    }
    return p3;
}

poly_head multiply(poly_head p1, poly_head p2)
{
    poly_head p3;
```

```
    p3.start = NULL;
    p3.num_terms = 0;

    poly_term *a=p1.start, *b;
    while(a!=NULL)
    {
        b = p2.start;
        while(b!=NULL)
        {
            term_insert(&p3, a->coeff*b->coeff, a->xpower+b->xpower);
            b = b->next;
        }
        a = a->next;
    }
    return p3;
}

int main()
{
    poly_head p1;
    p1.start = NULL;
    p1.num_terms = 0;
    new_poly(&p1);

    poly_head p2;
    p2.start = NULL;
    p2.num_terms = 0;
    new_poly(&p2);

    poly_head p3 = add(p1,p2);
    printf("P1 + P2 : \n");
    displayPolynomial(p3);
    printf("\n\n");
    poly_head p4 = multiply(p1,p2);
    printf("P1 * P2 : \n");
    displayPolynomial(p4);
}
/*
OUTPUT
Terms in Polynomial : 3
Enter The Polynomial :
Term 1 : 4 2
Term 2 : 5 1
Term 3 : 7 0

Terms in Polynomial : 2
Enter The Polynomial :
Term 1 : 1 2
Term 2 : 3 0
P1 + P2 :
(5)x^2 + (5)x^1 + (10)x^0


P1 * P2 :
(4)x^4 + (5)x^3 + (19)x^2 + (15)x^1 + (21)x^0
*/
```