```c
// Write program in C to sort a given set of integers using Quick Sort Algorithm

// Quick sort in C

#include <stdio.h>

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

int partition(int array[], int low, int high)
{

    int pivot = array[high];

    int i = (low - 1);

    for (int j = low; j < high; j++)
    {
        if (array[j] <= pivot)
        {
            i++;
            swap(&array[i], &array[j]);
        }
    }

    swap(&array[i + 1], &array[high]);

    return (i + 1);
}

void quickSort(int array[], int low, int high)
{
    if (low < high)
    {

        int pi = partition(array, low, high);
```

```
        quickSort(array, low, pi - 1);

        quickSort(array, pi + 1, high);
    }
}

int main()
{
    int a[10], n, i;

    printf("Ente the number of element in array\n");
    scanf("%d",&n);

    printf("Enter the element in array\n");
    for ( i = 0; i < n; i++)
    {
        scanf("%d", &a[i]);
    }

    quickSort(a, 0, n - 1);

    printf("Sorted array in ascending order: \n");

    for (int i = 0; i < n; ++i)
    {
        printf("%d  ", a[i]);
    }
    printf("\n");
}
// Write program in C to sort a given set of integers using
Merge Sort Algorithm.

#include <stdio.h>

void merge(int arr[], int p, int q, int r)
{

    int n1 = q - p + 1;
    int n2 = r - q;

    int L[n1], M[n2];
```

```
    for (int i = 0; i < n1; i++)
        L[i] = arr[p + i];
    for (int j = 0; j < n2; j++)
        M[j] = arr[q + 1 + j];

    int i, j, k;
    i = 0;
    j = 0;
    k = p;

    while (i < n1 && j < n2)
    {
        if (L[i] <= M[j])
        {
            arr[k] = L[i];
            i++;
        }
        else
        {
            arr[k] = M[j];
            j++;
        }
        k++;
    }

    while (i < n1)
    {
        arr[k] = L[i];
        i++;
        k++;
    }

    while (j < n2)
    {
        arr[k] = M[j];
        j++;
```

```
        k++;
    }
}

void mergeSort(int arr[], int l, int r)
{
    if (l < r)
    {

        int m = l + (r - l) / 2;

        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);
        merge(arr, l, m, r);
    }
}

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

int main()
{
    int a[10], n, i;

    printf("Ente the number of element in array\n");
    scanf("%d",&n);

    printf("Enter the element in array\n");
    for ( i = 0; i < n; i++)
    {
        scanf("%d", &a[i]);
    }

    mergeSort(a, 0, n - 1);
```

```
    printf("Sorted array: \n");
    printArray(a, n);
}
// Write a program in C to traverse the nodes of a graph using
Breadth First Search (BFS).

#include <stdio.h>

#include <conio.h>

int a[20][20], q[20], visited[20], n, i, j, f = 0, r = -1;

void bfs(int v)
{

    for (i = 1; i <= n; i++)

        if (a[v][i] && !visited[i])

            q[++r] = i;

    if (f <= r)
    {

        visited[q[f]] = 1;

        bfs(q[f++]);
    }
}

void main()
{

    // clrscr();

    int v;

    printf("Enter the number of vertices: ");

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

```
    for (i = 1; i <= n; i++)
    {

        q[i] = 0;

        visited[i] = 0;
    }

    printf("\nEnter graph data in matrix form:\n");

    for (i = 1; i <= n; i++)
    {

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

            scanf("%d", &a[i][j]);
        }
    }

    printf("Enter the starting vertex: ");

    scanf("%d", &v);

    bfs(v);

    printf("\nThe node which are reachable are:");

    for (i = 1; i <= n; i++)
    {

        if (visited[i])

            printf(" %d", i);

        else
        {

            printf("\nBFS is not possible. All nodes are not
reachable!");
```

```
            break;
        }
    }

    getch();
}
// Write a program in C to traverse the nodes of a graph using
Depth First Search (DFS).

#include <stdio.h>

#include <conio.h>

int a[20][20], reach[20], n;

void dfs(int v)

{

    int i;

    reach[v] = 1;

    for (i = 1; i <= n; i++)

        if (a[v][i] && !reach[i])

        {

            printf("\n%d->%d", v, i);

            dfs(i);
        }
}

void main()

{

    // clrscr();

    int i, j, count = 0;
```

```
    printf("\nEnter number of vertices: ");

    scanf("%d", &n);

    for (i = 1; i <= n; i++)

    {

        reach[i] = 0;

        for (j = 1; j <= n; j++)

            a[i][j] = 0;
    }

    printf("Enter the adjacency matrix:\n");

    for (i = 1; i <= n; i++)

        for (j = 1; j <= n; j++)

            scanf("%d", &a[i][j]);

    dfs(1);

    printf("\n");

    for (i = 1; i <= n; i++)

    {

        if (reach[i])

            count++;
    }

    if (count == n)

        printf("Graph is connected");

    else
```

```
        printf("Graph is not connected");

    getch();
}
/**
 * Add two polynomials
 * Using Linked List
 * @author Swashata
 * @for Dearest Froggie
 */

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

/**
 * The structure for the polynomial
 * This is a linked list with two variable
 * int coeff The Coefficient
 * int pow The power of x
 */
typedef struct link
{
    int coeff;
    int pow;
    struct link *next;
} poly;

/** The prototypes */
void createpoly(poly **);
void showpoly(poly *);
void addpoly(poly **, poly *, poly *);


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

        printf("\nCreate 1st expression\n");
```

```
        createpoly(&poly1);
        printf("\nStored the 1st expression");
        showpoly(poly1);

        printf("\nCreate 2nd expression\n");
        createpoly(&poly2);
        printf("\nStored the 2nd expression");
        showpoly(poly2);

        addpoly(&poly3, poly1, poly2);
        showpoly(poly3);

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


void createpoly(poly **node)
{
    int flag; // A flag to control the menu
    int coeff, pow;
    poly *tmp_node;                          // To hold the
temporary last address
    tmp_node = (poly *)malloc(sizeof(poly)); // create the first
node
    *node = tmp_node;                           // Store the
head address to the reference variable
    do
    {
        // Get the user data
        printf("\nEnter Coeff:");
        scanf("%d", &coeff);
        tmp_node->coeff = coeff;
        printf("\nEnter Pow:");
        scanf("%d", &pow);
        tmp_node->pow = pow;
        // Done storing user data

        // Now increase the Linked on user condition
        tmp_node->next = NULL;
```

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

/**
 * The show polynomial function
 * Prints the Polynomial in user readable format
 * @param poly * node The polynomial linked list
 * @return void
 */
void showpoly(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(" + ");
    }
}

/**
 * The polynomial add function
 * Adds two polynomial to a given variable
 * @param poly ** result Stores the result
 * @param poly * poly1 The first polynomial expression
 * @param poly * poly2 The second polynomial expression
 * @return void
```

```
 */
void addpoly(poly **result, poly *poly1, poly *poly2)
{
    poly *tmp_node; // Temporary storage for the linked list
    tmp_node = (poly *)malloc(sizeof(poly));
    tmp_node->next = NULL;
    *result = tmp_node; // Copy the head address to the result
linked list

    // Loop while both of the linked lists have value
    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;
        }

        // Grow the linked list on condition
        if (poly1 && poly2)
        {
            tmp_node->next = (poly *)malloc(sizeof(poly));
            tmp_node = tmp_node->next;
            tmp_node->next = NULL;
        }
    }

    // Loop while either of the linked lists has value
```

```
    while (poly1 || poly2)
    {
        // We have to create the list at beginning
        // As the last while loop will not create any unnecessary
node
        tmp_node->next = (poly *)malloc(sizeof(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");
}
```