1. Write a C Program to search elements in an array.

```
// C code to Search elements in array
#include <stdio.h>

int search(int arr[], int N, int x)
{
    int i;

    // iterate through all the element of array
    for (i = 0; i < N; i++)
        if (arr[i] == x)
            return i;
    return -1;
}

int main(void)
{
    int arr[] = { 9, 3, 2, 1, 10, 4 };
    int x = 10;
    int N = sizeof(arr) / sizeof(arr[0]);

    // Function Call
    int result = search(arr, N, x);

    if (result == -1) {
        printf("Element is not present in array");
    }
    else {
        printf("Element is present at index %d", result);
    }

    return 0;
}
```

2. Write a C Program to search elements in an array using Binary Search.

```
// C program to Search element
// in Array using Binary Search
#include <stdio.h>

int binarySearch(int arr[], int l, int r, int x)
{
    if (r >= l) {
        int mid = l + (r - l) / 2;

        // If the element is present at the middle
        // itself
```

```
        if (arr[mid] == x)
            return mid;

        // If element is smaller than mid, then
        // it can only be present in left subarray
        if (arr[mid] > x)
            return binarySearch(arr, l, mid - 1, x);

        // Else the element can only be present
        // in right subarray
        return binarySearch(arr, mid + 1, r, x);
    }

    return -1;
}

int main()
{
    int arr[] = { 11, 14, 19, 23, 40 };
    int n = sizeof(arr) / sizeof(arr[0]);
    int x = 40;
    int result = binarySearch(arr, 0, n - 1, x);
    if (result == -1) {
        printf("Element is not present in array");
    }
    else {
        printf("Element is present at index %d", result);
    }
    return 0;
}
```

3. Write a C Program to sort arrays using Bubble, Selection, and Insertion Sort.

```
// C Program to implement
// Sorting Algorithms
#include <stdio.h>

// A function to implement bubble sort
void bubble_sort(int* arr, int n)
{
    for (int j = 0; j < n - 1; j++) {

        // Last j elements are already in place
        for (int i = 0; i < n - j - 1; i++) {
            if (arr[i] > arr[i + 1]) {
                int temp = arr[i];
                arr[i] = arr[i + 1];
                arr[i + 1] = temp;
```

```
            }
        }
    }
}

// A function to implement swaping
void swap(int* xp, int* yp)
{
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}

// A function to implement selectionSort
void selectionSort(int arr[], int n)
{

    // One by one move boundary of unsorted subarray
    for (int i = 0; i < n - 1; i++) {
        // Find the minimum element in unsorted array
        int min_idx = i;
        for (int j = i + 1; j < n; j++)
            if (arr[j] < arr[min_idx])
                min_idx = j;

        // Swap the found minimum element
        // with the first element
        if (min_idx != i)
            swap(&arr[min_idx], &arr[i]);
    }
}

void insertionSort(int arr[], int n)
{

    for (int i = 1; i < n; i++) {
        int key = arr[i];
        int j = i - 1;

        // Move elements of arr that are
        // greater than key, to one position ahead
        // of their current position
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
```

```
}

int main()
{
    int arr1[] = { 9, 4, 3, 11, 1, 5 };
    int arr2[] = { 4, 3, 9, 1, 5, 11 };
    int arr3[] = { 5, 1, 11, 3, 4, 9 };
    int n = 6;

    printf("Non-Sorted array: ");
    for (int i = 0; i < n; i++)
        printf("%d ", arr1[i]);
    printf("\n");

    // sort array
    bubble_sort(arr1, n);

    // printing array
    printf("Sorted array using Bubble sort: ");
    for (int i = 0; i < n; i++)
        printf("%d ", arr1[i]);
    printf("\n");

    printf("Non-Sorted array: ");
    for (int i = 0; i < n; i++)
        printf("%d ", arr2[i]);
    printf("\n");

    // sort array
    insertionSort(arr2, n);

    // printing array
    printf("Sorted array using Insertion Sort: ");
    for (int i = 0; i < n; i++)
        printf("%d ", arr2[i]);
    printf("\n");

    printf("Non-Sorted array: ");
    for (int i = 0; i < n; i++)
        printf("%d ", arr3[i]);
    printf("\n");

    // sort array
    selectionSort(arr3, n);

    // printing array
    printf("Sorted array using Selection Sort: ");
    for (int i = 0; i < n; i++)
```

```
        printf("%d ", arr3[i]);
    printf("\n");

    return 0;
}
```

4. Write a C Program to sort arrays using Merge Sort.

```
// C program for
// Sorting array
// using Merge Sort
#include <stdio.h>

void merge(int arr[], int l, int m, int r)
{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;

    // create temperary arrays
    int L[n1], R[n2];

    // Copy data to arrays from L[] and R[]
    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];

    // Initial index of first ,second
    // and merged subarray respectively
    i = 0;
    j = 0;
    k = l;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        }
        else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    // Copy the remaining elements of L[]
    while (i < n1) {
        arr[k] = L[i];
```

```
        i++;
        k++;
    }

    // Copy the remaining elements of R[]
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}

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

        // calculating middle term
        int mid = l + (r - l) / 2;

        // divide to sort both halves
        mergeSort(arr, l, mid);
        mergeSort(arr, mid + 1, r);

        merge(arr, l, mid, r);
    }
}

int main()
{
    int arr[] = { 23, 9, 13, 15, 6, 7 };
    int n = sizeof(arr) / sizeof(arr[0]);

    // Printing orignal array
    printf("Given array:");
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");

    mergeSort(arr, 0, n - 1);

    // Printing sorted array
    printf("Sorted array :");
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");

    return 0;
}
```

5. Write a C Program to sort arrays using Quick Sort.

```
// C Program for
// sorting array using
// Quick sort
#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);

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

    // swap the pivot element with the greater element at i
    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);
    }
}

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

```
    printf("\n");
}

int main()
{
    int arr[] = { 28, 7, 20, 1, 10, 3 , 6 };

    int n = sizeof(arr) / sizeof(arr[0]);

    printf("Unsorted Array:");
    printArray(arr, n);

    quickSort(arr, 0, n - 1);

    printf("Sorted array :");
    printArray(arr, n);

    return 0;
}
```

6. Write a program to sort an array using pointers.

```
// C Program to implement
// sorting using pointers
#include <stdio.h>

// Function to sort the numbers using pointers
void sort(int n, int* ptr)
{
    int i, j;

    // Sort the numbers using pointers
    for (i = 0; i < n; i++) {

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

            if (*(ptr + j) < *(ptr + i)) {

                int temp = *(ptr + i);
                *(ptr + i) = *(ptr + j);
                *(ptr + j) = temp;
            }
        }
    }

    // print the numbers
    for (i = 0; i < n; i++)
        printf("%d ", *(ptr + i));
```

```
}

// Driver code
int main()
{
    int n = 5;
    int arr[] = { 13, 22, 7, 12, 4 };

    sort(n, arr);

    return 0;
}
```

7. Write a C program to Store Information about Students Using Structure

```
// C Program to Store
// Information about Students
// Using Structure
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Create the student structure
struct Student {
    char* name;
    int roll_number;
    int age;
};

// Driver code
int main()
{
    int n = 3;

    // Create the student's structure variable
    // with n Student's records
    struct Student student[n];

    // Get the students data
    student[0].roll_number = 1;
    student[0].name = "Geeks1";
    student[0].age = 10;

    student[1].roll_number = 2;
    student[1].name = "Geeks2";
    student[1].age = 11;

    student[2].roll_number = 3;
```

```
    student[2].name = "Geeks3";
    student[2].age = 13;

    // Printing  the Structers
    printf("Student Records:\n\n");
    for (int i = 0; i < n; i++) {
        printf("\tName : %s", student[i].name);
        printf("\tRoll Number : %d",
               student[i].roll_number);
        printf("\tAge : %d\n", student[i].age);
    }

    return 0;
}
```

8. Write a C Program To Add Two Complex Numbers Using Structures And Functions.

```
// C program to demonstrate
// addition of complex numbers
#include <stdio.h>

// define a structure for complex number
typedef struct complexNumber {
    int real;
    int img;
} complex;


complex add(complex x, complex y)
{
    // define a new complex number.
    complex add;

    // add similar type together
    add.real = x.real + y.real;
    add.img = x.img + y.img;

    return (add);
}

int main()
{

    // define three complex type numbers
    complex x, y, sum;

    // first complex number
    x.real = 4;
```

```
    x.img = 5;

    // second complex number
    y.real = 7;
    y.img = 11;

    // printing both complex numbers
    printf(" x = %d + %di\n", x.real, x.img);
    printf(" y = %d + %di\n", y.real, y.img);

    // call add(a,b) function and
    // pass complex numbers a & b
    // as an parameter.
    sum = add(x, y);

    // print result
    printf("\n sum = %d + %di", sum.real, sum.img);

    return 0;
}
```

9. Write a C Program to add Two Distance Given as Input in Feet and Inches

```
// C program for calculating sum of
// Distance in intches and feet
#include "stdio.h"

// Struct defined for the inch-feet system
struct InchFeet {
    int feet;
    float inch;
};

// Function to find the sum of all N
// set of Inch Feet distances
void findSum(struct InchFeet arr[], int N)
{

    // Variable to store sum
    int feet_sum = 0;
    float inch_sum = 0.0;

    int x;

    // Traverse the InchFeet array
    for (int i = 0; i < N; i++) {

        // Find the total sum of
```

```
        // feet and inch
        feet_sum += arr[i].feet;
        inch_sum += arr[i].inch;
    }

    // If inch sum is greater than 11
    // convert it into feet
    // as 1 feet = 12 inch
    if (inch_sum >= 12) {

        // Find integral part of inch_sum
        x = (int)inch_sum;

        // Delete the integral part x
        inch_sum -= x;

        // Add x%12 to inch_sum
        inch_sum += x % 12;

        // Add x/12 to feet_sum
        feet_sum += x / 12;
    }

    // Print the corresponding sum of
    // feet_sum and inch_sum
    printf("Feet Sum: %d\n", feet_sum);
    printf("Inch Sum: %.2f", inch_sum);
}

int main()
{
    struct InchFeet arr[]
        = { { 11, 5.1 }, { 13, 4.5 }, { 6, 8.1 } };

    int N = sizeof(arr) / sizeof(arr[0]);

    findSum(arr, N);

    return 0;
}
```

10. Write a C program to reverse a linked list iteratively

```
// C program to reverse a linked list iteratively
#include <stdio.h>
#include <stdlib.h>

/* Link list node */
```

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

/* Function to reverse the linked list */
static void reverse(struct Node** head_ref)
{
    struct Node* prev = NULL;
    struct Node* current = *head_ref;
    struct Node* next = NULL;
    while (current != NULL) {
        // Store next
        next = current->next;

        // Reverse current node's pointer
        current->next = prev;

        // Move pointers one position ahead.
        prev = current;
        current = next;
    }
    *head_ref = prev;
}

/* Function to push a node */
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;
}

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

/* Driver code*/
int main()
{
```

```
    /* Start with the empty list */
    struct Node* head = NULL;

    push(&head, 10);
    push(&head, 14);
    push(&head, 19);
    push(&head, 25);

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