1) **Write a C program to read the data from the file “employee.txt” which contains empno and empname and sort the data on names alphabetically (use strcmp) using Bubble Sort.**

Ans :_ employee.txt

```
1001 John

1002 Alice

1003 Bob

...#include &lt;stdio.h&gt;

#include &lt;string.h&gt;

#define MAX_EMPLOYEES 100

#define MAX_NAME_LENGTH 50

typedef struct {

int empno;

char empname[MAX_NAME_LENGTH];

} Employee;

void bubbleSort(Employee arr[], int n) {

int i, j;

Employee temp;

for (i = 0; i &lt; n-1; i++) {

for (j = 0; j &lt; n-i-1; j++) {

if (strcmp(arr[j].empname, arr[j+1].empname) &gt; 0) {

// Swap arr[j] and arr[j+1]

temp = arr[j];

arr[j] = arr[j+1];

arr[j+1] = temp;

}

}

}

}

int main() {
```

```
FILE *file;
Employee employees[MAX_EMPLOYEES];
int count = 0;
// Open the file in read mode
file = fopen(&quot;employee.txt&quot;, &quot;r&quot;);
if (file == NULL) {
printf(&quot;Could not open file employee.txt\n&quot;);
return 1;
}
// Read data from the file
while (fscanf(file, &quot;%d %s&quot;, &amp;employees[count].empno, employees[count].empname) != EOF) {
count++;
}
fclose(file);
// Sort the array using Bubble Sort
bubbleSort(employees, count);
// Print the sorted array
printf(&quot;Sorted employee data by names:\n&quot;);
for (int i = 0; i &lt; count; i++) {
printf(&quot;%d %s\n&quot;, employees[i].empno, employees[i].empname);
}
return 0;
}
```

2) **Write a C program to read the data from the file “person.txt” which contains person no and person age and sort the data on age in ascending order using insertion Sort**

Ans :- person.txt

```
1 25
2 30
3 22
...
```

```
#include &lt;stdio.h&gt;
#define MAX_PERSONS 100
typedef struct {
int personNo;
int personAge;
} Person;
void insertionSort(Person arr[], int n) {
int i, j;
Person key;
for (i = 1; i &lt; n; i++) {
key = arr[i];
j = i - 1;
// Move elements of arr[0..i-1], that are greater than key,
// to one position ahead of their current position
while (j &gt;= 0 &amp;&amp; arr[j].personAge &gt; key.personAge) {
arr[j + 1] = arr[j];
j = j - 1;
}
arr[j + 1] = key;
}
}
int main() {
FILE *file;
Person persons[MAX_PERSONS];
int count = 0;

// Open the file in read mode

file = fopen(&quot;person.txt&quot;, &quot;r&quot;);

if (file == NULL) {

printf(&quot;Could not open file person.txt\n&quot;);

return 1;

}

// Read data from the file

while (fscanf(file, &quot;%d %d&quot;, &amp;persons[count].personNo,
&amp;persons[count].personAge) !=

EOF) {

count++;

}

fclose(file);

// Sort the array using Insertion Sort

insertionSort(persons, count);

// Print the sorted array

printf(&quot;Sorted person data by age:\n&quot;);

for (int i = 0; i &lt; count; i++) {

printf(&quot;%d %d\n&quot;, persons[i].personNo, persons[i].personAge);

}

return 0;

}
```

3 **Write a program in C to accept 5 numbers from the user and sort the numbers in ascending order by using Merge sort.**

```
Ans :- #include &lt;stdio.h&gt;

// Function to merge two halves

void merge(int arr[], int l, int m, int r) {

int n1 = m - l + 1;

int n2 = r - m;

int L[n1], R[n2];

for (int i = 0; i &lt; n1; i++)
```

```
L[i] = arr[l + i];
for (int j = 0; j &lt; n2; j++)
R[j] = arr[m + 1 + j];
int i = 0;
int j = 0;
int k = l;
while (i &lt; n1 &amp;&amp; j &lt; n2) {
if (L[i] &lt;= R[j]) {
arr[k] = L[i];
i++;
} else {
arr[k] = R[j];
j++;
}
k++;
}
while (i &lt; n1) {
arr[k] = L[i];
i++;

k++;
}
while (j &lt; n2) {
arr[k] = R[j];
j++;
k++;
}
}
// MergeSort function
void mergeSort(int arr[], int l, int r) {
if (l &lt; r) {
```

```
int m = l + (r - l) / 2;
mergeSort(arr, l, m);
mergeSort(arr, m + 1, r);

// Merge the sorted halves
merge(arr, l, m, r);
}
}
int main() {
int arr[5];
// Accepting 5 numbers from the user
printf(&quot;Enter 5 numbers: \n&quot;);
for (int i = 0; i &lt; 5; i++) {
scanf(&quot;%d&quot;, &amp;arr[i]);
}
// Sorting the array using Merge sort
mergeSort(arr, 0, 4);
// Printing the sorted array
printf(&quot;Sorted array in ascending order: \n&quot;);
for (int i = 0; i &lt; 5; i++) {
printf(&quot;%d &quot;, arr[i]);
}
return 0;
}
```

3) **Write a C program to sort a random array of n integers by using Merge Sort algorithm in ascending order.**

Ans:- #include &lt;stdio.h&gt;

```
#include &lt;stdlib.h&gt;
#include &lt;time.h&gt;
void merge(int arr[], int l, int m, int r) {
```

```
int n1 = m - l + 1;
int n2 = r - m;
// Create temp arrays
int L[n1], R[n2];
// Copy data to temp arrays L[] and R[]
for (int i = 0; i &lt; n1; i++)
L[i] = arr[l + i];
for (int j = 0; j &lt; n2; j++)
R[j] = arr[m + 1 + j];
// Merge the temp arrays back into arr[l..r]
int i = 0;
int j = 0;
int k = l;
while (i &lt; n1 &amp;&amp; j &lt; n2) {
if (L[i] &lt;= R[j]) {
arr[k] = L[i];
i++;
} else {
arr[k] = R[j];
j++;
}
k++;
}
// Copy the remaining elements of L[], if there are any
while (i &lt; n1) {
arr[k] = L[i];
i++;
k++;
}
// Copy the remaining elements of R[], if there are any
while (j &lt; n2) {
```

```
arr[k] = R[j];

j++;

k++;

}

}

// MergeSort function

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

if (l &lt; r) {

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

// Sort first and second halves

mergeSort(arr, l, m);

mergeSort(arr, m + 1, r);

// Merge the sorted halves

merge(arr, l, m, r);

}

}

int main() {

int n;

// Asking user for the number of elements in the array

printf(&quot;Enter the number of elements in the array: &quot;);

scanf(&quot;%d&quot;, &amp;n);

// Dynamically allocating memory for the array

int* arr = (int*)malloc(n * sizeof(int));

if (arr == NULL) {

printf(&quot;Memory allocation failed\n&quot;);

return 1;

}

// Seeding the random number generator

srand(time(0));

// Generating random array of n integers

printf(&quot;Unsorted array: \n&quot;);
```

```
for (int i = 0; i &lt; n; i++) {
arr[i] = rand() % 100; // Random numbers between 0 and 99
printf(&quot;%d &quot;, arr[i]);
}
printf(&quot;\n&quot;);
// Sorting the array using Merge sort
mergeSort(arr, 0, n - 1);
// Printing the sorted array
printf(&quot;Sorted array in ascending order: \n&quot;);
for (int i = 0; i &lt; n; i++) {
printf(&quot;%d &quot;, arr[i]);
}
printf(&quot;\n&quot;);
// Freeing the allocated memory
free(arr);
return 0;
}
```

4) **Write a C program to create an array of integers. Accept a value from user and use linear search method to check whether the given number is present in array or not. Display proper message in output.**

```
Ans :- #include <stdio.h>
        int main() {
       int n, i, searchElement, found = 0;
 // Input the number of elements in the array
   printf("Enter the number of elements in the array: ");
   scanf("%d", &n);
 // Declare the array
   int arr[n];
 // Input the elements of the array
   printf("Enter %d elements:\n", n);
   for (i = 0; i < n; i++) {
      printf("Element %d: ", i + 1);
```

```
        scanf("%d", &arr[i]);
    }
 // Input the element to search for
    printf("Enter the number to search for: ");
    scanf("%d", &searchElement);
   // Linear search
    for (i = 0; i < n; i++) {
        if (arr[i] == searchElement) {
            found = 1;
            break;
        }
    }
  // Display the result
 if (found) {
        printf("The number %d is present in the array.\n", searchElement);
    } else {
        printf("The number %d is not present in the array.\n", searchElement);
    }

    return 0;
}
```

5) **Write a C program to accept n elements from user store it in an array. Accept a value from the user and use Non- recursive binary search method to check whether the value is present in array or not. Display proper message in output.**

```
Ans :- #include <stdio.h>
// Function to perform non-recursive binary search
int binarySearch(int arr[], int size, int target) {
    int left = 0;
    int right = size - 1;
while (left <= right) {
        int mid = left + (right - left) / 2;
```

```
  // Check if target is present at mid
        if (arr[mid] == target) {
            return mid;
        }
   // If target is greater, ignore left half
        if (arr[mid] < target) {
            left = mid + 1;
        } else {
            // If target is smaller, ignore right half
            right = mid - 1;
        }
    }
 return -1;
}
int main() {
    int n, i, target, index;
printf("Enter the number of elements: ");
    scanf("%d", &n);
 int arr[n];
    printf("Enter %d sorted elements:\n", n);
    for (i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
printf("Enter the value to search for: ");
    scanf("%d", &target);
    index = binarySearch(arr, n, target);
if (index != -1) {
        printf("Value %d found at index %d.\n", target, index);
    } else {
        printf("Value %d not found in the array.\n", target);
    }
 return 0;
}
```

6) Write a C program to implement a Doubly Circular linked list with following operations create() and display()

Ans :-

```
           #include <stdio.h>
         #include <stdlib.h>
      typedef struct Node {
  int data;
  struct Node* next;
  struct Node* prev;
} Node;
Node* head = NULL;
void create(int data) {
 Node* newNode = (Node*)malloc(sizeof(Node));
  newNode->data = data;
  newNode->next = newNode->prev = NULL;  // Initialize pointers
  if (head == NULL) {
    head = newNode;
    head->next = head->prev = head;
  } else {
    Node* last = head->prev;
    last->next = newNode;
    newNode->prev = last;
    newNode->next = head;
    head->prev = newNode;
  }
}
void display() {
  if (head == NULL) {
    printf("List is empty.\n");
    return;
```

```
   }
 Node* temp = head;
   do {
       printf("%d ", temp->data);
       temp = temp->next;
   } while (temp != head);
       printf("\n");
}
int main() {
   int n, data;
   printf("Enter number of elements to insert: ");
   scanf("%d", &n);
   for (int i = 0; i < n; i++) {
       printf("Enter element %d: ", i + 1);
       scanf("%d", &data);
       create(data);
   }
   printf("Doubly Circular Linked List: ");
   display();
  return 0;
}
```

7) **Write a C program to reverse the given string by using static and dynamic implementation of stack.**

Ans :_ Static Stack Implementation

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 10 // Maximum number of digits (change as needed)
typedef struct {
   int items[MAX];
   int top;
} Stack;
```

```
void initStack(Stack *s) {
   s->top = -1;
}
int isEmpty(Stack *s) {
   return s->top == -1;
}
int isFull(Stack *s) {
   return s->top == MAX - 1;
}
void push(Stack *s, int item) {
   if (!isFull(s)) {
      s->items[++(s->top)] = item;
   } else {
      printf("Stack overflow\n")   }    }
int pop(Stack *s) {
   if (!isEmpty(s)) {
      return s->items[(s->top)--];
   } else {
      printf("Stack underflow\n");
      return -1; // Error value
   }
}
void reverseNumberStatic(int number) {
   Stack s;
   initStack(&s);
   while (number > 0) {
      push(&s, number % 10);
      number /= 10;
   }
   int reversedNumber = 0;
   while (!isEmpty(&s)) {
        reversedNumber = reversedNumber * 10 + pop(&s);
    }
      printf("Reversed number (static stack): %d\n", reversedNumber);
}
int main() {
    int number;
    printf("Enter a number to reverse: ");
    scanf("%d", &number);

    reverseNumberStatic(number);
 return 0;
}
```

Dynamic Stack Implementation

```
#include <stdio.h>     #include <stdlib.h>
typedef struct Node {
    int data;
    struct Node *next;
} Node;
typedef struct {
    Node *top;
} Stack;
void initStack(Stack *s) {
    s->top = NULL;
}
int isEmpty(Stack *s) {
    return s->top == NULL;
}
void push(Stack *s, int item) {
    Node *newNode = (Node *)malloc(sizeof(Node));
    if (newNode != NULL) {
        newNode->data = item;
        newNode->next = s->top;
        s->top = newNode;
    } else {
        printf("Memory allocation failed\n");
    }
}
int pop(Stack *s) {
    if (!isEmpty(s)) {
        Node *temp = s->top;
        int item = temp->data;
        s->top = s->top->next;
        free(temp);
        return item;
    } else
  printf("Stack underflow\n");
        return -1; // Error value
    }
}
void reverseNumberDynamic(int number) {
    Stack s;
    initStack(&s);
    while (number > 0) {
        push(&s, number % 10);
        number /= 10;
    }
    int reversedNumber = 0;
    while (!isEmpty(&s)) {
        reversedNumber = reversedNumber * 10 + pop(&s);
    }   printf("Reversed number (dynamic stack): %d\n", reversedNumber);
}
int main() {
    int number;

    printf("Enter a number to reverse: ");

    scanf("%d", &number);

    reverseNumberDynamic(number);

  return 0;

}
```

8) a C program to reverse the given number by using static and dynamic implementation of stack Write.

```
Ans :_ #include <stdio.h>

          #define MAX 100

        int stack[MAX];  // Static stack

       int top = -1;    // Initialize top of stack

          void push(int n) {

    if (top < MAX - 1) {  // Check if stack is not full

 stack[++top] = n;

    }

}

int pop() {

    if (top >= 0) {  // Check if stack is not empty

        return stack[top--];

    }

    return -1;

}

int main() {

    int num, remainder;

  printf("Enter a number to reverse: ");

    scanf("%d", &num);

while (num != 0) {

        remainder = num % 10;  // Extract the last digit

        push(remainder);       // Push onto stack

        num /= 10;            // Remove last digit
    }
printf("Reversed number (Static Stack): ");
    // Pop elements from stack and print to reverse the number
    while (top >= 0) {
        printf("%d", pop());
    }
    printf("\n");
    return 0;
}
#include <stdio.h>
#include <stdlib.h>
typedef struct StackNode {
    int data;
    struct StackNode* next;
} StackNode;
StackNode* top = NULL;  // Initialize top of dynamic stack
void push(int n) {
    StackNode* newNode = (StackNode*)malloc(sizeof(StackNode));  // Allocate memory for new node
    newNode->data = n;  // Store data in new node
    newNode->next = top;  // Point to current top
    top = newNode;  // Update top to new node
}
int pop() {
    if (top != NULL) {  // Check if stack is not empty
        StackNode* temp = top;  // Temporary pointer to current top
        int data = top->data;  // Store data of top
        top = top->next;  // Move top to next node
        free(temp);  // Free memory of old top
        return data;  // Return popped data
    }
    return -1;  // Return -1 if stack is empty
```

```
}
int main() {
    int num, remainder;
  printf("Enter a number to reverse: ");
    scanf("%d", &num);
    while (num != 0) {
        remainder = num % 10;  // Extract the last digit
        push(remainder);      // Push onto stack
        num /= 10;            // Remove last digit
    }
   printf("Reversed number (Dynamic Stack): ");
      while (top != NULL) {
        printf("%d", pop());
    }   printf("\n");      return 0;          }
```

9) **Write a C program to Implement Dynamic implementation of Queue of integers with following operation: Initialize() b) insert() c) delete()**

Ans :-

```
          #include <stdio.h>
        #include <stdlib.h>
      typedef struct Node {
    int data;          // Store data
    struct Node* next;  // Pointer to the next node
} Node;
Node* front = NULL;
Node* rear = NULL;
void initialize() {
    front = rear = NULL;
    printf("Queue initialized.\n");
}
void insert(int value) {
    Node* newNode = (Node*)malloc(sizeof(Node));  // Allocate memory for new node
    if (newNode == NULL) {  // Check if memory allocation failed
```

```
      printf("Memory allocation failed.\n");
        return;
    }
    newNode->data = value;
    newNode->next = NULL;
    if (rear == NULL) {
        front = rear = newNode;
    } else {
        rear->next = newNode;
        rear = newNode;
    }
   printf("Inserted %d into the queue.\n", value);
}
void delete() {
    if (front == NULL) {  // Check if the queue is empty
        printf("Queue is empty, nothing to delete.\n");
        return;
    }

    Node* temp = front;  // Temporary pointer to hold the front node
    front = front->next;  // Move front to the next node

    // If front becomes NULL, set rear to NULL as well
    if (front == NULL) {
        rear = NULL;
    }

    printf("Deleted %d from the queue.\n", temp->data);
    free(temp);  // Free the memory of the deleted node
}
void display() {
    if (front == NULL) {  // Check if the queue is empty
        printf("Queue is empty.\n");
        return;
    }
 Node* temp = front;  // Temporary pointer to traverse the queue
    printf("Queue elements: ");
    while (temp != NULL) {  // Traverse until the end of the queue
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}
int main() {
    int choice, value;
 initialize();  // Initialize the queue

      while (1) {
        printf("\nQueue Operations:\n");
        printf("1. Insert\n2. Delete\n3. Display\n4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:  // Insert element
                printf("Enter value to insert: ");
                scanf("%d", &value);
                insert(value);
                break;
            case 2:  // Delete element
                delete();
                break;
            case 3:  // Display elements
```

```
                display();
                break;
            case 4:  // Exit
                printf("Exiting program.\n");
                exit(0);
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }

    return 0;
}
```

10) Write C programs to create and display the elements using Inorder traversal.

```
Ans :_  #include <stdio.h>
#include <stdlib.h>
typedef struct Node {
    int data;              // Store data
    struct Node* left;      // Pointer to left child
    struct Node* right;     // Pointer to right child
} Node;
Node* createNode(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));  // Allocate memory for new node
    newNode->data = data;
    newNode->left = newNode->right = NULL;       // Initialize children as NULL
    return newNode;
}
void inorderTraversal(Node* root) {
    if (root == NULL) {  // Base case: If the tree is empty
        return;
    }
      inorderTraversal(root->left);  // Traverse the left subtree
    printf("%d ", root->data);     // Print the data of the node
    inorderTraversal(root->right); // Traverse the right subtree
}
Node* insert(Node* root, int data) {
    if (root == NULL) {  // If the tree is empty, create a new node
        return createNode(data);
    }
  if (data < root->data) {
        root->left = insert(root->left, data);  // Insert in the left subtree
    } else if (data > root->data) {
        root->right = insert(root->right, data); // Insert in the right subtre  }
 return root;  // Return the root node after insertion
}
int main() {
    Node* root = NULL;  // Initialize the root of the tree
    int n, value;
    printf("Enter the number of elements to insert in the binary tree: ");
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        printf("Enter element %d: ", i + 1);
        scanf("%d", &value);
        root = insert(root, value);
    }
    printf("Inorder Traversal of the Binary Tree: ");
    inorderTraversal(root);
    printf("\n");
    return 0;
}
```

11) Write a C program to create binary search tree of integers and perform following operations: Pre order traversal Post order traversal

```
Ans :_  #include <stdio.h>
        #include <stdlib.h>
        typedef struct Node {
   int data;              // Store data
   struct Node* left;      // Pointer to left child
   struct Node* right;     // Pointer to right child
} Node;
Node* createNode(int data) {
   Node* newNode = (Node*)malloc(sizeof(Node));
   newNode->data = data;
   newNode->left = newNode->right = NULL;
   return newNode;  }
Node* insert(Node* root, int data) {
   if (root == NULL) {  // If the tree is empty, create a new node
      return createNode(data);
   }
   if (data < root->data) {
      root->left = insert(root->left, data);  // Insert in the left subtree
   } else if (data > root->data) {
      root->right = insert(root->right, data); // Insert in the right subtree
   }
      return root;  // Return the root node after insertion
}
void preorderTraversal(Node* root) {
   if (root == NULL) {  // Base case: If the tree is empty
      return;
   }
  printf("%d ", root->data);   // Print the data of the node
   preorderTraversal(root->left);  // Traverse the left subtree
```

```
    preorderTraversal(root->right); // Traverse the right subtree
}
void postorderTraversal(Node* root) {
    if (root == NULL) {  // Base case: If the tree is empty
        return;
    }
  postorderTraversal(root->left);  // Traverse the left subtree
    postorderTraversal(root->right); // Traverse the right subtree
    printf("%d ", root->data);       // Print the data of the node
}
int main() {
    Node* root = NULL;  // Initialize the root of the tree
    int n, value;

    printf("Enter the number of elements to insert in the binary tree: ");
    scanf("%d", &n);
  for (int i = 0; i < n; i++) {
        printf("Enter element %d: ", i + 1);
        scanf("%d", &value);
        root = insert(root, value);
    }
printf("Preorder Traversal of the Binary Search Tree: ");
    preorderTraversal(root);
    printf("\n");
    printf("Postorder Traversal of the Binary Search Tree: ");
    postorderTraversal(root);
    printf("\n");

    return 0;
}
```