1. Merge sort

```
#include <stdio.h>

void mergeSortedParts(int arr[], int left, int mid, int right) {
    int n1 = mid - left + 1, n2 = right - mid;
    int L[n1], R[n2];
    for (int i = 0; i < n1; i++)
        L[i] = arr[left + i];
    for (int j = 0; j < n2; j++)
        R[j] = arr[mid + 1 + j];
    int i = 0, j = 0, k = left;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j])
            arr[k++] = L[i++];
        else
            arr[k++] = R[j++];
    }
    while (i < n1)
        arr[k++] = L[i++];
    while (j < n2)
        arr[k++] = R[j++];
}

void mergeSort(int arr[], int left, int right) {
    if (left >= right)
        return;
    int mid = left + (right - left) / 2;
    mergeSort(arr, left, mid);
    mergeSort(arr, mid + 1, right);
```

```
    mergeSortedParts(arr, left, mid, right);
}

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

int main() {
    int len;
    printf("Enter number of elements: ");
    scanf("%d", &len);
    int arr[len];
    printf("Enter elements: ");
    for (int i = 0; i < len; i++)
        scanf("%d", &arr[i]);
    printf("Original array: ");
    printArray(arr, len);
    mergeSort(arr, 0, len - 1);
    printf("Sorted array: ");
    printArray(arr, len);
    return 0;
}
```

2. Bubble sort

```
#include <stdio.h>

void bubl(int arr[], int n) {
   if (n == 1)
      return;
   for (int i = 0; i < n - 1; i++) {
      if (arr[i] > arr[i + 1]) {
         int temp = arr[i];
         arr[i] = arr[i + 1];
         arr[i + 1] = temp;
      }
   }
   bubli(arr, n - 1);
}

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

int main() {

   int len;
   printf("Enter number of elements: ");
   scanf("%d", &len);
   int arr[len];
   printf("Enter elements: ");
   for (int i = 0; i < len; i++)
      scanf("%d", &arr[i]);
   printf("Original array: ");
   printArray(arr, len);
   bubli(arr, len);
   printf("Sorted array: ");
   printArray(arr, len);

   return 0;
}
```

3. Queue

```
#include <stdio.h>
#define SIZE 10

struct queue {
   int data[SIZE];
   int front, rear;
};

void push(struct queue* q) {
   if (((q->front) + 1) % SIZE == q->rear) {
      printf("Overflow\n");
      return;
   }
   if (q->rear == -1) // Initialize the queue
      q->rear++;
   q->front = (q->front + 1) % SIZE;
   int val;
   printf("Enter value to push: ");
   scanf("%d", &val);
   q->data[q->front] = val;
}

void del(struct queue* q) {
   if (q->rear == -1) {
      printf("Underflow\n");
      return;
   }
   printf("Deleted %d\n", q->data[q->rear]);
```

```
    if (q->rear == q->front) { // Queue becomes empty
        q->rear = -1;
        q->front = -1;
    } else
        q->rear = (q->rear + 1) % SIZE;
}

void front(struct queue* q) {
    (q->front == -1) ? printf("Queue is empty\n") : printf("Front element: %d\n", q->data[q->front]);
}

void rear(struct queue* q) {
    (q->rear == -1) ? printf("Queue is empty\n") : printf("Rear element: %d\n", q->data[q->rear]);
}

void traverse(struct queue q) {
    if (q.rear == -1) {
        printf("Queue is empty\n");
        return;
    }
    printf("Queue elements: ");
    int i = q.rear;
    while (i != q.front) {
        printf("%d ", q.data[i]);
        i = (i + 1) % SIZE;
    }
    printf("%d\n", q.data[q.front]);
}
```

```
void printUtility(struct queue* q) {
    printf("\n- - - - - - - - - - - \n");
    printf("1. Push Element\n");
    printf("2. Remove Element\n");
    printf("3. Display Front Element\n");
    printf("4. Display Rear Element\n");
    printf("5. Display All Elements\n");
    printf("[Press any other key to exit]\n");
    int ch;
    scanf("%d", &ch);
    switch (ch) {
    case 1:
        push(q);
        break;
    case 2:
        del(q);
        break;
    case 3:
        front(q);
        break;
    case 4:
        rear(q);
        break;
    case 5:
        traverse(*q);
        break;
    default:
        return;
```

```
    }
    printUtility(q);
}

int main() {
    struct queue q;
    q.front = -1;
    q.rear = -1;
    printUtility(&q);
    return 0;
}
```

4. Stack

```
#include <stdio.h>
#define SIZE 10

struct stack {
   int arr[SIZE];
   int top;
};

void push(struct stack* s) {
   if (s->top == SIZE - 1) {
      printf("Stack Overflow\n");
      return;
   }
   int val;
   printf("Enter value to push: ");
   scanf("%d", &val);
   s->arr[++s->top] = val;
}

void pop(struct stack* s) {
   if (s->top == -1) {
      printf("Stack Underflow\n");
      return;
   }
   printf("Pop Element Successful\n");
   s->top--;
}

void top(struct stack* s) {
   (s->top == -1) ? printf("Stack Empty\n") : printf("%d\n", s->arr[s->top]);
}

void printUtility(struct stack* s) {
   printf("\n- - - - - - - - - - \n");
   printf("1. Push Element\n");
   printf("2. Pop Element\n");
   printf("3. Display Top\n");
   printf("[ Press any other key to exit ]\n");
   int ch;
   scanf("%d", &ch);
   switch (ch) {
```

```
    case 1:
        push(s);
        break;

    case 2:
        pop(s);
        break;

    case 3:
        top(s);
        break;

    default:
        return;
    }
    printUtility(s);
}

int main() {
    struct stack s;
    s.top = -1;
    printUtility(&s);
    return 0;
}
```

5. Singly Linked list

```
#include <stdio.h>
#include <stdlib.h>
struct ll {
   int data;
   struct ll* next;
};
void printList(struct ll* head) {
   if (head == NULL) {
      printf("Empty list\n");
      return;
   }
   struct ll* temp = head;
   while (temp != NULL) {
      printf("%d -> ", temp->data);
      temp = temp->next;
   }
   printf("NULL\n");
}
void insertHead(struct ll** head) {
   int value;
   printf("Enter value to insert at beginning: ");
   scanf("%d", &value);
   struct ll* newNode = (struct ll*)malloc(sizeof(struct ll));
   newNode->data = value;
   newNode->next = *head;
   *head = newNode;
   printf("Node inserted at the beginning.\n");
}
void insertTail(struct ll* head) {
   int value;
   printf("Enter value to insert at end: ");
   scanf("%d", &value);
   struct ll* newNode = (struct ll*)malloc(sizeof(struct ll));
   newNode->data = value;
   newNode->next = NULL;
   if (head == NULL) {
      head = newNode;
      return;
   }
   struct ll* temp = head;
   while (temp->next != NULL) {
      temp = temp->next;
   }
```

```
    temp->next = newNode;
    printf("Node inserted at the end.\n");
}
void deleteHead(struct ll** head) {
    if (*head == NULL) {
        printf("List is empty. No node to delete.\n");
        return;
    }
    struct ll* temp = *head;
    *head = (*head)->next;
    free(temp);
    printf("Node deleted from the beginning.\n");
}
void deleteTail(struct ll* head) {
    if (head == NULL) {
        printf("List is empty. No node to delete.\n");
        return;
    }
    struct ll* temp = head;
    struct ll* prev = NULL;

    if (temp->next == NULL) {
        free(temp);
        head = NULL;
        printf("Node deleted from the end.\n");
        return;
    }
    while (temp->next != NULL) {
        prev = temp;
        temp = temp->next;
    }
    prev->next = NULL;
    free(temp);
    printf("Node deleted from the end.\n");
}
void sortList(struct ll* head) {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }
    struct ll* i = head;
    struct ll* j = NULL;
    int temp;
    while (i != NULL) {
```

```
        j = i->next;
        while (j != NULL) {
            if (i->data > j->data) {
                temp = i->data;
                i->data = j->data;
                j->data = temp;
            }
            j = j->next;
        }
        i = i->next;
    }
    printf("List sorted.\n");
}
void printUtility(struct ll* head) {
    printf("\n- - - - - - - - - - - \n");
    printf("1. Traverse\n");
    printf("2. Insert at Beginning\n");
    printf("3. Insert at End\n");
    printf("4. Delete from Beginning\n");
    printf("5. Delete from End\n");
    printf("6. Sort list\n");
    printf("[ Press any other key to exit ]\n");
    int ch;
    scanf("%d", &ch);
    switch (ch) {
    case 1:
        printList(head);
        break;
    case 2:
        insertHead(&head);
        break;
    case 3:
        insertTail(head);
        break;
    case 4:
        deleteHead(&head);
        break;
    case 5:
        deleteTail(head);
        break;
    case 6:
        sortList(head);
        break;
    default:
```

```
        return;
    }
    printUtility(head);
}
int main() {
    struct ll* head = NULL;
    printUtility(head);
    return 0;
}
```

6. Tree

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

// Define a structure for the binary tree node
struct Node {
   int data;
   struct Node* left;
   struct Node* right;
};

// Function to create a new node
struct Node* createNode(int data) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = data;
   newNode->left = NULL;
   newNode->right = NULL;
   return newNode;
}

// Function to insert a node in the binary tree
struct Node* insertNode(struct Node* root, int data) {
   if (root == NULL) {
      return createNode(data);
   }

   if (data < root->data) {
      root->left = insertNode(root->left, data);
   } else if (data > root->data) {
      root->right = insertNode(root->right, data);
   }

   return root;
}

// Function to search for an element in the tree and print messages
int searchElement(struct Node* root, int key) {
   if (root == NULL) {
      printf("Visiting element: NULL - Element not found\n");
      return 0;
   }

   printf("Visiting element: %d - ", root->data);
   if (root->data == key) {
```

```
        printf("Element found\n");
        return 1;
    } else if (key < root->data) {
        printf("Element not found, moving left\n");
        return searchElement(root->left, key);
    } else {
        printf("Element not found, moving right\n");
        return searchElement(root->right, key);
    }
}

// In-order traversal (Left, Root, Right)
void inorderTraversal(struct Node* root) {
    if (root == NULL) return;
    inorderTraversal(root->left);
    printf("%d ", root->data);
    inorderTraversal(root->right);
}

// Pre-order traversal (Root, Left, Right)
void preorderTraversal(struct Node* root) {
    if (root == NULL) return;
    printf("%d ", root->data);
    preorderTraversal(root->left);
    preorderTraversal(root->right);
}

// Post-order traversal (Left, Right, Root)
void postorderTraversal(struct Node* root) {
    if (root == NULL) return;
    postorderTraversal(root->left);
    postorderTraversal(root->right);
    printf("%d ", root->data);
}

int main() {
    struct Node* root = NULL;

    // Inserting nodes into the binary tree
    root = insertNode(root, 50);
    insertNode(root, 30);
    insertNode(root, 20);
    insertNode(root, 40);
    insertNode(root, 70);
```

```
    insertNode(root, 60);
    insertNode(root, 80);

    int key;
    printf("Enter element to search: ");
    scanf("%d", &key);

    // Search for the element in the tree
    if (!searchElement(root, key)) {
        printf("The element %d is not present in the tree.\n", key);
    }

    // Display tree traversals
    printf("\nIn-order traversal: ");
    inorderTraversal(root);
    printf("\n");

    printf("Pre-order traversal: ");
    preorderTraversal(root);
    printf("\n");

    printf("Post-order traversal: ");
    postorderTraversal(root);
    printf("\n");

    return 0;
}
```

7. Structure

```
#include <stdio.h>
struct student {
char firstName[50];
int roll;
float marks;
} s[5];
int main()
{
int i;
printf("Enter information of students:\n");
// storing information
for (i = 0; i < 5; ++i) {
s[i].roll = i + 1;
printf("\nFor roll number%d,\n", s[i].roll);
printf("Enter first name: ");
scanf("%s", s[i].firstName);
printf("Enter marks: ");
scanf("%f", &s[i].marks);
}
printf("Displaying Information:\n\n");
// displaying information
for (i = 0; i < 5; ++i) {
printf("\nRoll number: %d\n", i + 1);
printf("First name: ");
puts(s[i].firstName);
printf("Marks: %.1f", s[i].marks);
printf("\n");
}
return 0;
}
```