```
//program for single linked list----------------------------------------------

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

struct Node{
  int value;
  struct Node* next;
};

typedef struct Node n;

n *head = NULL;



n* createNode(int val){
    n* newnode;
    newnode = (n*)malloc(sizeof(n));
    newnode->next = NULL;
    newnode->value = val;
    return newnode;
}

int insertAtBeg(int val){
    n* newnode = createNode(val);
    if (head == NULL){
        head = newnode;
        return 0;
```

```
    }
    newnode->next = head;
    head = newnode;
    return 0;
    }


int insertAtEnd(int val){
    n* newnode = createNode(val);
    if (head == NULL){
        head = newnode;
        return 0;
    }
    n* temp = head;

    while (temp->next != NULL )
    {
        temp = temp->next;
    }
    temp->next = newnode;
    return 0;
    }



int deleteAtBeg(){

    if (head == NULL){
        printf("I am Empty");
```

```
    return 0;
  }
  n* temp = head;
  head = head->next;
  free(temp);

  return 0;
  }

int deleteAtPos(int pos){

  if (head == NULL){
    printf("I am Empty");
    return 0;
  }

  n* temp = head;

  for (int i = 1 ; i < pos-1 && temp != NULL; i++){
    temp = temp->next;
  }
  n* temp1 = temp->next;
  temp->next = temp->next->next;
  free(temp1);

  return 0;
  }
```

```
int deleteAtEnd(){

   if (head == NULL){
      printf("I am Empty");
      return 0;
   }

   if (head->next == NULL){
      deleteAtBeg();
      return 0;
   }

   n* temp = head;

   while ( temp->next->next != NULL){
      temp = temp->next;
   }
   n* temp1 = temp->next;
   temp->next = NULL;
   free(temp1);

   return 0;
   }


int insertAtPos(int val , int pos){
   n* newnode = createNode(val);
   if (pos == 1){
```

```
        insertAtBeg(val);
        return 0;
    }
    n *temp = head;
    for (int i = 1 ; i < pos-1 && temp != NULL; i++){
        temp = temp->next;
    }
    newnode->next = temp->next;
    temp->next = newnode;
    return 0;
    }


int display (){
    n *temp = head;

    if(temp == NULL){
        printf("I am empty");
    }
    while(temp != NULL){
        printf("%d->",temp->value);
        temp = temp->next;
    }
    return 0;
}

int main() {
    insertAtBeg(30);
    insertAtBeg(350);
```

```
    insertAtBeg(10);
    insertAtBeg(40);
    insertAtPos(40,2);
    deleteAtPos(3);
    deleteAtBeg();
    insertAtEnd(99);
    deleteAtEnd();

    display();
    return 0;
}


//program for doubly linked list----------------------------------------------------

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

struct Node{
  int value;
  struct Node* next;
  struct Node* prev;

};

typedef struct Node n;

n *head = NULL;
```

```
n* createNode(int val){
   n* newnode;
   newnode = (n*)malloc(sizeof(n));
   newnode->next = NULL;
   newnode->prev = NULL;
   newnode->value = val;
   return newnode;
}

int insertAtBeg(int val){
   n* newnode = createNode(val);
   if (head == NULL){
      head = newnode;
      return 0;
   }
   newnode->next = head;
   head->prev = newnode;
   head = newnode;
   return 0;
   }

int insertAtPos(int val , int pos){
   n* newnode = createNode(val);
   if (pos == 1){
      insertAtBeg(val);
      return 0;
   }
   n *temp = head;
```

```
    for (int i = 1 ; i < pos-1 && temp != NULL; i++){
        temp = temp->next;
    }
    newnode->next = temp->next;
    temp->next = newnode;
    newnode->prev = temp;
    newnode->next->prev = newnode;
    return 0;
    }


int insertAtEnd(int val){
    n* newnode = createNode(val);
    if (head == NULL){
        head = newnode;
        return 0;
    }
    n* temp = head;

    while (temp->next != NULL )
    {
        temp = temp->next;
    }
    temp->next = newnode;
    newnode->prev = temp;
    return 0;
    }
```

```
int deleteAtBeg(){

   if (head == NULL){
      printf("I am Empty");
      return 0;
   }
   n* temp = head;
   head = head->next;
   free(temp);

   return 0;
   }

int deleteAtPos(int pos){

   if (head == NULL){
      printf("I am Empty");
      return 0;
   }

   n* temp = head;

   for (int i = 1 ; i < pos-1 && temp != NULL; i++){
      temp = temp->next;
   }
   n* temp1 = temp->next;
   temp->next = temp->next->next;
   temp->next->prev = temp;
```

```
    free(temp1);

    return 0;
    }



int deleteAtEnd(){

    if (head == NULL){
        printf("I am Empty");
        return 0;
    }

    if (head->next == NULL){
        deleteAtBeg();
        return 0;
    }

    n* temp = head;

    while ( temp->next->next != NULL){
        temp = temp->next;
    }
    n* temp1 = temp->next;
    temp->next = NULL;
    free(temp1);

    return 0;
```

```
    }




int display (){
    n *temp = head;

    if(temp == NULL){
        printf("I am empty");
    }
    while(temp != NULL){
        printf("%d->",temp->value);
        temp = temp->next;
    }
    return 0;
}

int main() {
    insertAtBeg(30);
    insertAtBeg(350);
    insertAtBeg(10);
    insertAtBeg(40);
    insertAtPos(40,2);
    deleteAtPos(3);
    deleteAtBeg();
    insertAtEnd(99);
    deleteAtEnd();
```

```
    display();
    return 0;
}


// circular single lined list----------------------------------------------
#include <stdio.h>
#include <stdlib.h>

struct Node {
    int value;
    struct Node* next;
};

typedef struct Node Node;

Node *head = NULL;


Node* createNode(int val) {
    Node* newnode = (Node*)malloc(sizeof(Node));
    newnode->value = val;
    newnode->next = NULL;
    return newnode;
}

void insertAtBeg(int val) {
    Node* newnode = createNode(val);
```

```
    if (head == NULL) {
        head = newnode;
        newnode->next = head;
    } else {
        newnode->next = head->next;
        head->next = newnode;
    }
}

void insertAtPos(int val, int pos) {
    Node* newnode = createNode(val);
    if (pos == 1) {
        insertAtBeg(val);
    } else {
        Node* temp = head;
        for (int i = 1; i < pos - 1 && temp->next != head; i++) {
            temp = temp->next;
        }
        newnode->next = temp->next;
        temp->next = newnode;
    }
}

void insertAtEnd(int val) {
    Node* newnode = createNode(val);
    if (head == NULL) {
        head = newnode;
        newnode->next = head;
    } else {
```

```
        Node* temp = head;
        while (temp->next != head) {
            temp = temp->next;
        }
        temp->next = newnode;
        newnode->next = head;
    }
}

void deleteAtBeg() {
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }
    Node* temp = head;
    if (head->next == head) {
        head = NULL;
    } else {
        while (temp->next != head) {
            temp = temp->next;
        }
        temp->next = head->next;
        head = head->next;
    }
    free(temp);
}

void deleteAtPos(int pos) {
    if (head == NULL) {
```

```
        printf("List is empty\n");
        return;
    }
    Node* temp = head;
    Node* prev = NULL;
    if (pos == 1) {
        deleteAtBeg();
    } else {
        for (int i = 1; i < pos && temp->next != head; i++) {
            prev = temp;
            temp = temp->next;
        }
        if (temp == head) {
            printf("Invalid position\n");
            return;
        }
        prev->next = temp->next;
        free(temp);
    }
}

void deleteAtEnd() {
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }
    Node* temp = head;
    Node* prev = NULL;
    while (temp->next != head) {
```

```
        prev = temp;
        temp = temp->next;
    }
    if (temp == head) {
        head = NULL;
    } else {
        prev->next = head;
    }
    free(temp);
}

void display() {
    Node* temp = head;
    if (temp == NULL) {
        printf("List is empty\n");
        return;
    }
    printf("Circular Linked List: ");
    do {
        printf("%d->", temp->value);
        temp = temp->next;
    } while (temp != head);
    printf("\n");
}

int main() {
    insertAtBeg(30);
    insertAtBeg(350);
    insertAtBeg(10);
```

```
    insertAtBeg(40);
    insertAtPos(40, 2);
    deleteAtPos(3);
    deleteAtBeg();
    insertAtEnd(99);
    deleteAtEnd();

    display();
    return 0;
}



// circualar doubly linked list ----------------------------------------------
#include <stdio.h>
#include <stdlib.h>

struct Node{
  int value;
  struct Node* next;
};

typedef struct Node n;

n *head = NULL;

n* createNode(int val){
    n* newnode;
    newnode = (n*)malloc(sizeof(n));
```

```
    newnode->next = NULL;
    newnode->value = val;
    return newnode;
}

void insertAtBeg(int val){
    n* newnode = createNode(val);
    if (head == NULL){
        head = newnode;
        return;
    }
    newnode->next = head;
    head = newnode;
}

void display (){
    n *temp = head;

    while(temp != NULL){
        printf("%d->",temp->value);
        temp = temp->next;
    }
    printf("NULL\n");
}

void insertAtEnd(int val){
    n* newnode = createNode(val);
    if (head == NULL){
        head = newnode;
```

```
        return;
    }
    n* temp = head;
    while(temp->next != NULL){
        temp = temp->next;
    }
    temp->next = newnode;
}

void deleteAtBeg(){
    if (head == NULL){
        printf("List is empty. Cannot delete.\n");
        return;
    }
    n* temp = head;
    head = head->next;
    free(temp);
}

void deleteAtEnd(){
    if (head == NULL){
        printf("List is empty. Cannot delete.\n");
        return;
    }
    n* temp = head;
    n* prev = NULL;
    while(temp->next != NULL){
        prev = temp;
        temp = temp->next;
```

```
    }
    if (prev != NULL){
        prev->next = NULL;
        free(temp);
    } else {
        free(head);
        head = NULL;
    }
}

int main() {
    insertAtBeg(10);
    insertAtBeg(30);
    insertAtBeg(20);
    insertAtBeg(11);
    printf("Initial list:\n");
    display();

    printf("\nAfter inserting 25 at the end:\n");
    insertAtEnd(25);
    display();

    printf("\nAfter deleting from the beginning:\n");
    deleteAtBeg();
    display();

    printf("\nAfter deleting from the end:\n");
    deleteAtEnd();
    display();
```

```
   return 0;
}



//stack----------------------------------------------------------------------



#include <stdio.h>
#define MAX 50
#include <stdbool.h>

int Arr[MAX];
int tos = -1;




bool isEmpty(){
   return (tos == -1);
}

bool isFull(){

   return (tos == MAX-1);
}



int push(int val){
   if(isFull()){
```

```
        printf("The stack is full");
        return -1;
    }
    tos++;
    Arr[tos] = val;
}
int pop (){
       if(isEmpty()){
        printf("The stack is empty");
        return -1;
    }
    tos--;
}


void display(){
    for (int i = 0 ; i <= tos ; i++){
        printf("%d\n",Arr[i]);
    }
    return;
}

int main() {
    push(20);
    push(60);
    pop();
    push(220);
    push(10);
    display();
```

```
    return 0;
}



//queue linear-------------------------------------------------------------------------

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

#define MAX_SIZE 100

struct Queue {
    int front, rear, size;
    int array[MAX_SIZE];
};

struct Queue* createQueue() {
    struct Queue* queue = (struct Queue*)malloc(sizeof(struct Queue));
    queue->front = queue->size = 0;
    queue->rear = -1; // Rear is initialized to -1 for a linear queue
    return queue;
}

int isFull(struct Queue* queue) {
    return (queue->size == MAX_SIZE);
}

int isEmpty(struct Queue* queue) {
    return (queue->size == 0);
```

```
}

void enqueue(struct Queue* queue, int item) {
    if (isFull(queue)) {
        printf("Queue is full.\n");
        return;
    }
    queue->rear++;
    queue->array[queue->rear] = item;
    queue->size++;
    printf("%d enqueued to queue.\n", item);
}

int dequeue(struct Queue* queue) {
    if (isEmpty(queue)) {
        printf("Queue is empty.\n");
        return -1;
    }
    int item = queue->array[queue->front];
    queue->front++;
    queue->size--;
    return item;
}

int front(struct Queue* queue) {
    if (isEmpty(queue)) {
        printf("Queue is empty.\n");
        return -1;
    }
```

```
    return queue->array[queue->front];
}

int rear(struct Queue* queue) {
    if (isEmpty(queue)) {
        printf("Queue is empty.\n");
        return -1;
    }
    return queue->array[queue->rear];
}

void display(struct Queue* queue) {
    if (isEmpty(queue)) {
        printf("Queue is empty.\n");
        return;
    }
    printf("Queue elements are: ");
    int i;
    for (i = queue->front; i <= queue->rear; i++) {
        printf("%d ", queue->array[i]);
    }
    printf("\n");
}

int main() {
    struct Queue* queue = createQueue();

    enqueue(queue, 10);
    enqueue(queue, 20);
```

```
    enqueue(queue, 30);
    enqueue(queue, 40);

    printf("%d dequeued from queue.\n", dequeue(queue));

    printf("Front item is %d.\n", front(queue));
    printf("Rear item is %d.\n", rear(queue));

    display(queue);

    return 0;
}



//queue  circular-------------------------------------------------------------------------

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

#define MAX_SIZE 100

struct Queue {
    int front, rear, size;
    int array[MAX_SIZE];
};

struct Queue* createQueue() {
    struct Queue* queue = (struct Queue*)malloc(sizeof(struct Queue));
    queue->front = queue->size = 0;
```

```
    queue->rear = MAX_SIZE - 1;
    return queue;
}

int isFull(struct Queue* queue) {
    return (queue->size == MAX_SIZE);
}

int isEmpty(struct Queue* queue) {
    return (queue->size == 0);
}

void enqueue(struct Queue* queue, int item) {
    if (isFull(queue)) {
        printf("Queue is full.\n");
        return;
    }
    queue->rear = (queue->rear + 1) % MAX_SIZE;
    queue->array[queue->rear] = item;
    queue->size++;
    printf("%d enqueued to queue.\n", item);
}

int dequeue(struct Queue* queue) {
    if (isEmpty(queue)) {
        printf("Queue is empty.\n");
        return -1;
    }
    int item = queue->array[queue->front];
```

```
    queue->front = (queue->front + 1) % MAX_SIZE;
    queue->size--;
    return item;
}

int front(struct Queue* queue) {
    if (isEmpty(queue)) {
        printf("Queue is empty.\n");
        return -1;
    }
    return queue->array[queue->front];
}

int rear(struct Queue* queue) {
    if (isEmpty(queue)) {
        printf("Queue is empty.\n");
        return -1;
    }
    return queue->array[queue->rear];
}

void display(struct Queue* queue) {
    if (isEmpty(queue)) {
        printf("Queue is empty.\n");
        return;
    }
    printf("Queue elements are: ");
    int i;
    for (i = queue->front; i <= queue->rear; i++) {
```

```
        printf("%d ", queue->array[i]);
    }
    printf("\n");
}

int main() {
    struct Queue* queue = createQueue();

    enqueue(queue, 10);
    enqueue(queue, 20);
    enqueue(queue, 30);
    enqueue(queue, 40);

    printf("%d dequeued from queue.\n", dequeue(queue));



    display(queue);

    return 0;
}
```

//---

```
#include <stdio.h>

// Bubble sort
void bubbleSort(int arr[], int n) {
   int i, j, temp;
   for (i = 0; i < n - 1; i++) {
      for (j = 0; j < n - i - 1; j++) {
         if (arr[j] > arr[j + 1]) {
            temp = arr[j];
            arr[j] = arr[j + 1];
            arr[j + 1] = temp;
         }
      }
   }
}

// Insertion sort
void insertionSort(int arr[], int n) {
   int i, key, j;
   for (i = 1; i < n; i++) {
      key = arr[i];
      j = i - 1;
      while (j >= 0 && arr[j] > key) {
         arr[j + 1] = arr[j];
         j = j - 1;
      }
      arr[j + 1] = key;
   }
}
```

```
// Selection sort
void selectionSort(int arr[], int n) {
    int i, j, minIndex, temp;
    for (i = 0; i < n - 1; i++) {
        minIndex = i;
        for (j = i + 1; j < n; j++) {
            if (arr[j] < arr[minIndex]) {
                minIndex = j;
            }
        }
        temp = arr[minIndex];
        arr[minIndex] = arr[i];
        arr[i] = temp;
    }
}

int main() {
    int arr[] = {34, 12, 89, 45, 27};
    int n = sizeof(arr) / sizeof(arr[0]);

    // Bubble Sort
    bubbleSort(arr, n);
    printf("Bubble Sorted Array: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n\n");
```

```
    // Insertion Sort
    insertionSort(arr, n);
    printf("Insertion Sorted Array: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n\n");

    // Selection Sort
    selectionSort(arr, n);
    printf("Selection Sorted Array: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n\n");

    return 0;
}
```

```
//----------------------recursive sorting ----------------------------------

#include <stdio.h>

// Merge Sort
void merge(int arr[], int low, int mid, int high) {
    int i, j, k;
    int n1 = mid - low + 1;
```

```
int n2 = high - mid;
int Left[n1], Right[n2];
for (i = 0; i < n1; i++)
   Left[i] = arr[low + i];
for (j = 0; j < n2; j++)
   Right[j] = arr[mid + 1 + j];
i = 0;
j = 0;
k = low;
while (i < n1 && j < n2) {
   if (Left[i] <= Right[j]) {
      arr[k] = Left[i];
      i++;
   } else {
      arr[k] = Right[j];
      j++;
   }
   k++;
}
while (i < n1) {
   arr[k] = Left[i];
   i++;
   k++;
}
while (j < n2) {
   arr[k] = Right[j];
   j++;
   k++;
}
```

```
}

void mergeSort(int arr[], int low, int high) {
    if (low < high) {
        int mid = low + (high - low) / 2;
        mergeSort(arr, low, mid);
        mergeSort(arr, mid + 1, high);
        merge(arr, low, mid, high);
    }
}

// Quick Sort
int partition(int arr[], int low, int high) {
    int pivot = arr[high];
    int i = (low - 1);
    for (int j = low; j <= high - 1; j++) {
        if (arr[j] < pivot) {
            i++;
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }
    int temp = arr[i + 1];
    arr[i + 1] = arr[high];
    arr[high] = temp;
    return (i + 1);
}
```

```
void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

// Heap Sort (unchanged from iterative version)
void heapify(int arr[], int n, int i) {
    int largest = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;
    if (left < n && arr[left] > arr[largest])
        largest = left;

    if (right < n && arr[right] > arr[largest])
        largest = right;
    if (largest != i) {
        int temp = arr[i];
        arr[i] = arr[largest];
        arr[largest] = temp;
        heapify(arr, n, largest);
    }
}

void heapSort(int arr[], int n) {
    for (int i = n / 2 - 1; i >= 0; i--)
        heapify(arr, n, i);
```

```
    for (int i = n - 1; i >= 0; i--) {
        int temp = arr[0];
        arr[0] = arr[i];
        arr[i] = temp;
        heapify(arr, i, 0);
    }
}

int main() {
    int arr[] = {34, 12, 89, 45, 27};
    int n = sizeof(arr) / sizeof(arr[0]);

    // Merge Sort
    mergeSort(arr, 0, n - 1);
    printf("Merge Sorted Array: ");
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n\n");

    // Quick Sort
    quickSort(arr, 0, n - 1);
    printf("Quick Sorted Array: ");
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n\n");

    // Heap Sort
    heapSort(arr, n);
    printf("Heap Sorted Array: ");
```

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

    return 0;
}
```

----------searching----------

```
#include <stdio.h>

// Linear Search
int linearSearch(int arr[], int n, int target) {
    for (int i = 0; i < n; i++) {
        if (arr[i] == target) {
            return i;
        }
    }
    return -1; // Element not found
}

// Binary Search
int binarySearch(int arr[], int low, int high, int target) {
    while (low <= high) {
        int mid = low + (high - low) / 2;
        if (arr[mid] == target) {
            return mid;
        } else if (arr[mid] < target) {
            low = mid + 1;
```

```
        } else {
            high = mid - 1;
        }
    }
    return -1; // Element not found
}

int main() {
    int arr[] = {12, 27, 34, 45, 89};
    int n = sizeof(arr) / sizeof(arr[0]);
    int target;
    for (int i = 0; i < n;i++){
        printf("%d ",arr[i]);
    }
    printf("\n");
    printf("Enter the element to search: ");
    scanf("%d", &target);

    // Linear Search
    int linearIndex = linearSearch(arr, n, target);
    if (linearIndex != -1) {
        printf("Linear Search: Element found at index %d\n", linearIndex);
    } else {
        printf("Linear Search: Element not found\n");
    }

    // Binary Search
    int binaryIndex = binarySearch(arr, 0, n - 1, target);
    if (binaryIndex != -1) {
```

```
        printf("Binary Search: Element found at index %d\n", binaryIndex);
    } else {
        printf("Binary Search: Element not found\n");
    }

    return 0;
}
```