Q1:Write a program to perform various operations in the list:

- **Insertion**
- **Deletion**
- **Display**

```
#include <stdio.h>

#define MAX 100

int main() {
    int arr[MAX], n = 0, choice, pos, value, i;

    while (1) {
        printf("\n---- List Operations ----\n");
        printf("1. Insert\n");
        printf("2. Delete\n");
        printf("3. Display\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1: // Insertion
                if (n == MAX) {
                    printf("List is full. Cannot insert.\n");
```

```
    } else {
        printf("Enter position to insert (1 to %d): ", n + 1);
        scanf("%d", &pos);
        printf("Enter value to insert: ");
        scanf("%d", &value);

        if (pos < 1 || pos > n + 1) {
            printf("Invalid position!\n");
        } else {
            for (i = n - 1; i >= pos - 1; i--) {
                arr[i + 1] = arr[i];
            }
            arr[pos - 1] = value;
            n++;
            printf("Inserted successfully!\n");
        }
    }
    break;

case 2: // Deletion
    if (n == 0) {
        printf("List is empty. Nothing to delete.\n");
    } else {
        printf("Enter position to delete (1 to %d): ", n);
```

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

            if (pos < 1 || pos > n) {
                printf("Invalid position!\n");
            } else {
                for (i = pos - 1; i < n - 1; i++) {
                    arr[i] = arr[i + 1];
                }
                n--;
                printf("Deleted successfully!\n");
            }
        }
        break;

    case 3: // Display
        if (n == 0) {
            printf("List is empty.\n");
        } else {
            printf("List elements: ");
            for (i = 0; i < n; i++) {
                printf("%d ", arr[i]);
            }
            printf("\n");
        }
```

```
            break;

        case 4:
            printf("Exiting program...\n");
            return 0;

        default:
            printf("Invalid choice! Try again.\n");
      }
   }
}
```

```
---- List Operations ----
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter position to insert (1 to 1): 1
Enter value to insert: 1
Inserted successfully!

---- List Operations ----
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3
List elements: 1

---- List Operations ----
1. Insert
2. Delete
3. Display
```

Q2:Write a program to find the arithmetic operations on matrices:

- **Sum and Subtraction**
- **Product of 2 matrices**
- **Transpose of a matrix**

```
#include <stdio.h>


int main() {
    int a[10][10], b[10][10], c[10][10];
    int r1, c1, r2, c2;
    int i, j, k, choice;

    while (1) {
        printf("\n---- Matrix Operations ----\n");
        printf("1. Sum of Two Matrices\n");
        printf("2. Subtraction of Two Matrices\n");
        printf("3. Product of Two Matrices\n");
        printf("4. Transpose of a Matrix\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {

            case 1:  // Sum
                printf("Enter rows and columns of matrices: ");
```

```
    scanf("%d %d", &r1, &c1);

    printf("Enter Matrix A:\n");
    for (i = 0; i < r1; i++)
        for (j = 0; j < c1; j++)
            scanf("%d", &a[i][j]);

    printf("Enter Matrix B:\n");
    for (i = 0; i < r1; i++)
        for (j = 0; j < c1; j++)
            scanf("%d", &b[i][j]);

    printf("Sum of matrices:\n");
    for (i = 0; i < r1; i++) {
        for (j = 0; j < c1; j++) {
            c[i][j] = a[i][j] + b[i][j];
            printf("%d ", c[i][j]);
        }
        printf("\n");
    }
    break;

case 2:  // Subtraction
    printf("Enter rows and columns of matrices: ");
```

```
    scanf("%d %d", &r1, &c1);

    printf("Enter Matrix A:\n");
    for (i = 0; i < r1; i++)
        for (j = 0; j < c1; j++)
            scanf("%d", &a[i][j]);

    printf("Enter Matrix B:\n");
    for (i = 0; i < r1; i++)
        for (j = 0; j < c1; j++)
            scanf("%d", &b[i][j]);

    printf("Subtraction of matrices (A - B):\n");
    for (i = 0; i < r1; i++) {
        for (j = 0; j < c1; j++) {
            c[i][j] = a[i][j] - b[i][j];
            printf("%d ", c[i][j]);
        }
        printf("\n");
    }
    break;

case 3:  // Product
    printf("Enter rows and columns of Matrix A: ");
```

```
scanf("%d %d", &r1, &c1);

printf("Enter rows and columns of Matrix B: ");
scanf("%d %d", &r2, &c2);

if (c1 != r2) {
    printf("Matrix multiplication not possible.\n");
    break;
}

printf("Enter Matrix A:\n");
for (i = 0; i < r1; i++)
    for (j = 0; j < c1; j++)
        scanf("%d", &a[i][j]);

printf("Enter Matrix B:\n");
for (i = 0; i < r2; i++)
    for (j = 0; j < c2; j++)
        scanf("%d", &b[i][j]);

// Initialize product matrix
for (i = 0; i < r1; i++)
    for (j = 0; j < c2; j++)
        c[i][j] = 0;
```

```
            // Multiply
            for (i = 0; i < r1; i++)
                for (j = 0; j < c2; j++)
                    for (k = 0; k < c1; k++)
                        c[i][j] += a[i][j] * b[k][j];  // mistake here? Actually should be: a[i][k] *
b[k][j]


            printf("Product of matrices:\n");
            for (i = 0; i < r1; i++) {
                for (j = 0; j < c2; j++)
                    printf("%d ", c[i][j]);
                printf("\n");
            }
            break;


        case 4:  // Transpose
            printf("Enter rows and columns of Matrix: ");
            scanf("%d %d", &r1, &c1);


            printf("Enter Matrix:\n");
            for (i = 0; i < r1; i++)
                for (j = 0; j < c1; j++)
                    scanf("%d", &a[i][j]);
```

```
            printf("Transpose of Matrix:\n");
            for (i = 0; i < c1; i++) {
                for (j = 0; j < r1; j++)
                    printf("%d ", a[j][i]);
                printf("\n");
            }
            break;

        case 5:
            return 0;

        default:
            printf("Invalid choice! Try again.\n");
    }
  }
}
```

```
---- Matrix Operations ----
1. Sum of Two Matrices
2. Subtraction of Two Matrices
3. Product of Two Matrices
4. Transpose of a Matrix
5. Exit
Enter your choice: 1
Enter rows and columns of matrices: 2 3
Enter Matrix A:
1 2 3
4 5 6
Enter Matrix B:
6 7 8
2 4 5
Sum of matrices:
7 9 11
6 9 11

---- Matrix Operations ----
1. Sum of Two Matrices
2. Subtraction of Two Matrices
3. Product of Two Matrices
4. Transpose of a Matrix
```

Q3:Write a Program to sort the list using:

- **Bubble Sort**
- **Quick Sort**
- **Insertion sort**
- **Merge Sort**
- **Heap Sort**

Also, find the comparison on the basis of time complexity.

```
#include <stdio.h>


void bubble(int a[], int n) {
    int i, j, t;
    for(i=0;i<n-1;i++)
        for(j=0;j<n-i-1;j++)
            if(a[j] > a[j+1]) { t=a[j]; a[j]=a[j+1]; a[j+1]=t; }
}


void insertion(int a[], int n) {
    int i, j, key;
    for(i=1;i<n;i++) {
        key = a[i];
        j = i-1;
        while(j>=0 && a[j] > key) {
            a[j+1] = a[j];
            j--;
        }
        a[j+1] = key;
    }
```

```
}

void quick(int a[], int low, int high) {
    int i = low, j = high, pivot = a[(low+high)/2], t;
    while(i <= j) {
        while(a[i] < pivot) i++;
        while(a[j] > pivot) j--;
        if(i <= j) { t=a[i]; a[i]=a[j]; a[j]=t; i++; j--; }
    }
    if(low < j) quick(a, low, j);
    if(i < high) quick(a, i, high);
}

void mergeArr(int a[], int l, int m, int h) {
    int i=l, j=m+1, k=0, b[100];
    while(i<=m && j<=h)
        b[k++] = (a[i] < a[j]) ? a[i++] : a[j++];
    while(i<=m) b[k++] = a[i++];
    while(j<=h) b[k++] = a[j++];
    for(i=l, k=0; i<=h; i++, k++) a[i] = b[k];
}

void merge(int a[], int l, int h) {
    if(l < h) {
```

```
        int m = (l+h)/2;
        merge(a,l,m);
        merge(a,m+1,h);
        mergeArr(a,l,m,h);
    }
}

void heapify(int a[], int n, int i) {
    int l=2*i+1, r=2*i+2, largest=i, t;
    if(l<n && a[l]>a[largest]) largest=l;
    if(r<n && a[r]>a[largest]) largest=r;
    if(largest!=i) { t=a[i]; a[i]=a[largest]; a[largest]=t; heapify(a,n,largest); }
}

void heap(int a[], int n) {
    int i, t;
    for(i=n/2-1;i>=0;i--) heapify(a,n,i);
    for(i=n-1;i>0;i--) {
        t=a[0]; a[0]=a[i]; a[i]=t;
        heapify(a,i,0);
    }
}

void print(int a[], int n) {
```

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

int main() {
    int a[50], n, ch, i;

    printf("Enter n: ");
    scanf("%d", &n);
    printf("Enter elements: ");
    for(i=0;i<n;i++) scanf("%d",&a[i]);

    printf("\n1.Bubble 2.Quick 3.Insertion 4.Merge 5.Heap\nChoice: ");
    scanf("%d", &ch);

    if(ch==1) bubble(a,n);
    else if(ch==2) quick(a,0,n-1);
    else if(ch==3) insertion(a,n);
    else if(ch==4) merge(a,0,n-1);
    else if(ch==5) heap(a,n);
    else printf("Invalid!");

    printf("\nSorted List: ");
    print(a,n);
```

```
    printf("\n\nTime Complexity:\n");
    printf("Bubble Sort    : O(n^2)\n");
    printf("Insertion Sort : O(n^2)\n");
    printf("Quick Sort     : O(n log n)\n");
    printf("Merge Sort     : O(n log n)\n");
    printf("Heap Sort      : O(n log n)\n");

    return 0;
}
```

```
Enter n: 4
Enter elements: 3
1
2
5

1.Bubble 2.Quick 3.Insertion 4.Merge 5.Heap
Choice: 2

Sorted List: 1 2 3 5

Time Complexity:
Bubble Sort    : O(n^2)
Insertion Sort : O(n^2)
Quick Sort     : O(n log n)
Merge Sort     : O(n log n)
Heap Sort      : O(n log n)
```

Q4:Write a program to search the element using:

- **Linear Search**
- **Binary Search**

Also, find the comparison on the basis of time complexity.

```
#include <stdio.h>


int linearSearch(int a[], int n, int x) {
    int i;
    for(i = 0; i < n; i++)
        if(a[i] == x)
            return i;
    return -1;
}


int binarySearch(int a[], int n, int x) {
    int l = 0, r = n - 1, mid;
    while(l <= r) {
        mid = (l + r) / 2;
        if(a[mid] == x) return mid;
        else if(a[mid] < x) l = mid + 1;
        else r = mid - 1;
    }
    return -1;
}


int main() {
```

```
    int a[50], n, i, x, choice, pos;

    printf("Enter number of elements: ");
    scanf("%d", &n);

    printf("Enter sorted elements: ");
    for(i = 0; i < n; i++)
        scanf("%d", &a[i]);

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

    printf("\n1. Linear Search\n2. Binary Search\nChoice: ");
    scanf("%d", &choice);

    if(choice == 1) {
        pos = linearSearch(a, n, x);
        if(pos == -1) printf("Element not found.\n");
        else printf("Element found at position %d\n", pos+1);
    }
    else if(choice == 2) {
        pos = binarySearch(a, n, x);
        if(pos == -1) printf("Element not found.\n");
        else printf("Element found at position %d\n", pos+1);
```

```
    }
    else {
        printf("Invalid choice.\n");
    }

    printf("\nTime Complexity:\n");
    printf("Linear Search : O(n)\n");
    printf("Binary Search : O(log n)\n");

    return 0;
}
```

```
Enter number of elements: 2
Enter sorted elements: 1
2
Enter element to search: 2

1. Linear Search
2. Binary Search
Choice: 2
Element found at position 2

Time Complexity:
Linear Search : O(n)
Binary Search : O(log n)


=== Code Execution Successful ===
```

Q5:Write a program to perform all operations:

-For the Stack using an Array.

-For the Queue using an Array.

- For the Circular Queue using an Array.

```
#include <stdio.h>
#define SIZE 5

int stack[SIZE], top = -1;
int queue[SIZE], front = -1, rear = -1;
int cqueue[SIZE], cfront = -1, crear = -1;

/* ------ STACK OPERATIONS ------ */
void push(int x) {
   if(top == SIZE-1) printf("Stack Overflow\n");
   else stack[++top] = x;
}

void pop() {
   if(top == -1) printf("Stack Underflow\n");
   else printf("Popped: %d\n", stack[top--]);
}

void displayStack() {
   if(top == -1) printf("Stack Empty\n");
   else {
      printf("Stack: ");
      for(int i=0;i<=top;i++) printf("%d ", stack[i]);
      printf("\n");
   }
}
```

```
/* ------ QUEUE OPERATIONS ------ */
void enqueue(int x) {
    if(rear == SIZE-1) printf("Queue Full\n");
    else {
        if(front == -1) front = 0;
        queue[++rear] = x;
    }
}

void dequeue() {
    if(front == -1 || front > rear) printf("Queue Empty\n");
    else printf("Deleted: %d\n", queue[front++]);
}

void displayQueue() {
    if(front == -1 || front > rear) printf("Queue Empty\n");
    else {
        printf("Queue: ");
        for(int i=front;i<=rear;i++) printf("%d ", queue[i]);
        printf("\n");
    }
}

/* ------ CIRCULAR QUEUE OPERATIONS ------ */
void cenqueue(int x) {
    if((cfront == 0 && crear == SIZE-1) || (crear + 1 == cfront))
        printf("Circular Queue Full\n");
    else {
        if(cfront == -1) cfront = 0;
        crear = (crear + 1) % SIZE;
        cqueue[crear] = x;
    }
}
```

```
void cdequeue() {
    if(cfront == -1) printf("Circular Queue Empty\n");
    else {
        printf("Deleted: %d\n", cqueue[cfront]);
        if(cfront == crear) cfront = crear = -1;
        else cfront = (cfront + 1) % SIZE;
    }
}

void displayCQueue() {
    if(cfront == -1) printf("Circular Queue Empty\n");
    else {
        int i = cfront;
        printf("Circular Queue: ");
        while(1) {
            printf("%d ", cqueue[i]);
            if(i == crear) break;
            i = (i + 1) % SIZE;
        }
        printf("\n");
    }
}

/* ------ MAIN ------ */
int main() {
    push(10); push(20); push(30);
    displayStack();
    pop();
    displayStack();

    printf("\n");

    enqueue(5); enqueue(15); enqueue(25);
```

```
    displayQueue();
    dequeue();
    displayQueue();

    printf("\n");

    cenqueue(1); cenqueue(2); cenqueue(3);
    displayCQueue();
    cdequeue();
    displayCQueue();

    return 0;
}
```

```
Stack: 10 20 30
Popped: 30
Stack: 10 20

Queue: 5 15 25
Deleted: 5
Queue: 15 25

Circular Queue: 1 2 3
Deleted: 1
Circular Queue: 2 3


=== Code Execution Successful ===
```

Q6: Write a program to evaluate the Postfix Notation.

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

int stack[50], top = -1;

void push(int x) {
   stack[++top] = x;
}

int pop() {
   return stack[top--];
}

int main() {
   char exp[50];
   int i, a, b;

   printf("Enter postfix expression: ");
   scanf("%s", exp);

   for(i = 0; exp[i] != '\0'; i++) {
      if(isdigit(exp[i])) {
         push(exp[i] - '0');
      }
      else {
         b = pop();
         a = pop();
         switch(exp[i]) {
            case '+': push(a + b); break;
            case '-': push(a - b); break;
            case '*': push(a * b); break;
```

```
            case '/': push(a / b); break;
        }
    }
  }

  printf("Result = %d\n", pop());
  return 0;
}
```

```
Enter postfix expression: 23*54*+9-
Result = 17


=== Code Execution Successful ===
```

Q7:Write a program to convert infix notation into Postfix Notation.

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

char stack[50];
int top = -1;

void push(char x) { stack[++top] = x; }
char pop() { return stack[top--]; }

int priority(char x) {
   if (x == '(') return 0;
   if (x == '+' || x == '-') return 1;
   if (x == '*' || x == '/') return 2;
   return 0;
}

int main() {
   char infix[50], postfix[50];
   int i, j = 0;

   printf("Enter Infix Expression: ");
   scanf("%s", infix);

   for (i = 0; infix[i] != '\0'; i++) {
      char ch = infix[i];

      if (isalnum(ch))
         postfix[j++] = ch;
      else if (ch == '(')
         push(ch);
      else if (ch == ')') {
```

```
            while (stack[top] != '(')
                postfix[j++] = pop();
            pop();
        }
        else {
            while (top != -1 && priority(stack[top]) >= priority(ch))
                postfix[j++] = pop();
            push(ch);
        }
    }

    while (top != -1)
        postfix[j++] = pop();

    postfix[j] = '\0';

    printf("Postfix: %s", postfix);
    return 0;
}
```

```
Enter Infix Expression: (A+B)*C
Postfix: AB+C*

=== Code Execution Successful ===
```

Q8:Write a program to perform the operations of the linked list:

- **Insertion**
- **Deletion**
- **Display**

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

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

struct Node* head = NULL;

void insert(int x) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = x;
    newNode->next = head;
    head = newNode;
}

void delete(int x) {
    struct Node *temp = head, *prev = NULL;

    if (temp != NULL && temp->data == x) {
        head = temp->next;
        free(temp);
        return;
    }

    while (temp != NULL && temp->data != x) {
        prev = temp;
        temp = temp->next;
```

```
    }

    if (temp == NULL) return;

    prev->next = temp->next;
    free(temp);
}

void display() {
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
}

int main() {
    insert(10);
    insert(20);
    insert(30);
    printf("List after insertion: ");
    display();

    delete(20);
    printf("\nList after deletion: ");
    display();

    return 0;
}
```

```
List after insertion: 30 20 10
List after deletion: 30 10

=== Code Execution Successful ===
```

Q9:Write a program to perform the operations of the Circular linked list:

- **Insertion**
- **Deletion**
- **Display**

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

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

struct Node* head = NULL;

void insert(int x) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = x;

    if (head == NULL) {
        head = newNode;
        newNode->next = head;
    } else {
        struct Node* temp = head;
        while (temp->next != head)
            temp = temp->next;
        temp->next = newNode;
        newNode->next = head;
    }
}

void delete(int x) {
    if (head == NULL) return;
```

```
    struct Node *temp = head, *prev = NULL;

    if (temp->data == x && temp->next == head) {
        head = NULL;
        free(temp);
        return;
    }

    if (temp->data == x) {
        while (temp->next != head)
            temp = temp->next;
        temp->next = head->next;
        free(head);
        head = temp->next;
        return;
    }

    prev = head;
    temp = head->next;

    while (temp != head && temp->data != x) {
        prev = temp;
        temp = temp->next;
    }

    if (temp->data == x) {
        prev->next = temp->next;
        free(temp);
    }
}

void display() {
    if (head == NULL) return;
```

```
    struct Node* temp = head;
    do {
        printf("%d ", temp->data);
        temp = temp->next;
    } while (temp != head);
}

int main() {
    insert(10);
    insert(20);
    insert(30);

    printf("Circular List after insertion: ");
    display();

    delete(20);
    printf("\nCircular List after deletion: ");
    display();

    return 0;
}
```

```
Circular List after insertion: 10 20 30
Circular List after deletion: 10 30

=== Code Execution Successful ===
```

Q10:Write a program to perform all operations: -

- **For Stack using Linked List.**
- **For Queue using Linked List**
- **For Circular Queue using Linked List**

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

struct Node { int data; struct Node* next; };
struct Node *top=NULL, *front=NULL, *rear=NULL;
struct Node *frontC=NULL, *rearC=NULL;

/* STACK */
void push(int x){
   struct Node* n=malloc(sizeof(struct Node));
   n->data=x; n->next=top; top=n;
}
void pop(){
   if(!top) return;
   struct Node* t=top; top=top->next; free(t);
}
void displayStack(){
   struct Node* t=top;
   while(t){ printf("%d ",t->data); t=t->next; }
}

/* QUEUE */
void enqueue(int x){
   struct Node* n=malloc(sizeof(struct Node));
   n->data=x; n->next=NULL;
   if(!rear) front=rear=n;
   else{ rear->next=n; rear=n; }
}
void dequeue(){
```

```
    if(!front) return;
    struct Node* t=front;
    front=front->next;
    if(!front) rear=NULL;
    free(t);
}
void displayQueue(){
    struct Node* t=front;
    while(t){ printf("%d ",t->data); t=t->next; }
}

/* CIRCULAR QUEUE */
void enqueueC(int x){
    struct Node* n=malloc(sizeof(struct Node));
    n->data=x;
    if(!frontC){
        frontC=rearC=n;
        rearC->next=frontC;
    } else {
        rearC->next=n;
        rearC=n;
        rearC->next=frontC;
    }
}
void dequeueC(){
    if(!frontC) return;
    if(frontC==rearC){
        free(frontC);
        frontC=rearC=NULL;
        return;
    }
    struct Node* t=frontC;
    frontC=frontC->next;
    rearC->next=frontC;
```

```
    free(t);
}
void displayC(){
    if(!frontC) return;
    struct Node* t=frontC;
    do{
        printf("%d ",t->data);
        t=t->next;
    } while(t!=frontC);
}

int main(){
    int ch, val;

    while(1){
        printf("\n\n1.Push(Stack)\n2.Pop(Stack)\n3.Display Stack");
        printf("\n4.Enqueue(Queue)\n5.Dequeue(Queue)\n6.Display Queue");
        printf("\n7.Enqueue(Circular Queue)\n8.Dequeue(CQ)\n9.Display CQ");
        printf("\n10.Exit\nEnter choice: ");
        scanf("%d",&ch);

        if(ch==10) break;

        switch(ch){
            case 1: printf("Enter value: "); scanf("%d",&val); push(val); break;
            case 2: pop(); break;
            case 3: displayStack(); break;

            case 4: printf("Enter value: "); scanf("%d",&val); enqueue(val); break;
            case 5: dequeue(); break;
            case 6: displayQueue(); break;

            case 7: printf("Enter value: "); scanf("%d",&val); enqueueC(val); break;
            case 8: dequeueC(); break;
```

```
            case 9: displayC(); break;
        }
    }
    return 0;
}
```

```
4.Enqueue(Queue)
5.Dequeue(Queue)
6.Display Queue
7.Enqueue(Circular Queue)
8.Dequeue(CQ)
9.Display CQ
10.Exit
Enter choice: 1
Enter value: 2


1.Push(Stack)
2.Pop(Stack)
3.Display Stack
4.Enqueue(Queue)
5.Dequeue(Queue)
6.Display Queue
7.Enqueue(Circular Queue)
8.Dequeue(CQ)
9.Display CQ
10.Exit
Enter choice: 3
2
```

Q11:Write a program to perform concatenation of two linked lists.

```
#include <stdio.h>

#include <stdlib.h>


struct Node {

    int data;

    struct Node* next;

};


struct Node* createNode(int x) {

    struct Node* n = malloc(sizeof(struct Node));

    n->data = x;

    n->next = NULL;

    return n;

}


void insert(struct Node** head, int x) {

    struct Node* n = createNode(x);

    n->next = *head;

    *head = n;

}


struct Node* concatenate(struct Node* a, struct Node* b) {

    if (a == NULL) return b;
```

```
    struct Node* temp = a;
    while (temp->next != NULL)
        temp = temp->next;
    temp->next = b;
    return a;
}

void display(struct Node* head) {
    while (head) {
        printf("%d ", head->data);
        head = head->next;
    }
}

int main() {
    struct Node *list1 = NULL, *list2 = NULL;

    insert(&list1, 30);
    insert(&list1, 20);
    insert(&list1, 10);

    insert(&list2, 60);
    insert(&list2, 50);
    insert(&list2, 40);
```

```
    printf("List 1: ");
    display(list1);
    printf("\nList 2: ");
    display(list2);

    list1 = concatenate(list1, list2);

    printf("\nConcatenated List: ");
    display(list1);

    return 0;
}
```

```
List 1: 10 20 30
List 2: 40 50 60
Concatenated List: 10 20 30 40 50 60

=== Code Execution Successful ===
```

Q12:Write a program to perform the operations of the Double linked list:

- **Insertion**
- **Deletion**
- **Display**

```
#include <stdio.h>

#include <stdlib.h>


struct Node {

    int data;

    struct Node *prev, *next;

};


struct Node* head = NULL;


void insert(int x) {

    struct Node* n = malloc(sizeof(struct Node));

    n->data = x;

    n->prev = NULL;

    n->next = head;


    if (head != NULL)

        head->prev = n;


    head = n;

}
```

```
void delete(int x) {
    struct Node* temp = head;

    while (temp && temp->data != x)
        temp = temp->next;

    if (!temp) return;

    if (temp->prev)
        temp->prev->next = temp->next;
    else
        head = temp->next;

    if (temp->next)
        temp->next->prev = temp->prev;

    free(temp);
}

void display() {
    struct Node* temp = head;
    while (temp) {
        printf("%d ", temp->data);
```

```
        temp = temp->next;
    }
}

int main() {
    insert(10);
    insert(20);
    insert(30);

    printf("Doubly Linked List after insertion: ");
    display();

    delete(20);
    printf("\nDoubly Linked List after deletion: ");
    display();

    return 0;
}
```

```
Doubly Linked List after insertion: 30 20 10
Doubly Linked List after deletion: 30 10

=== Code Execution Successful ===
```

Q13:Write a program to perform tree traversal methods.

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

struct Node {
    int data;
    struct Node *left, *right;
};

struct Node* create(int x) {
    struct Node* n = malloc(sizeof(struct Node));
    n->data = x;
    n->left = n->right = NULL;
    return n;
}

void inorder(struct Node* root) {
    if (!root) return;
    inorder(root->left);
    printf("%d ", root->data);
    inorder(root->right);
}
```

```
void preorder(struct Node* root) {
    if (!root) return;
    printf("%d ", root->data);
    preorder(root->left);
    preorder(root->right);
}

void postorder(struct Node* root) {
    if (!root) return;
    postorder(root->left);
    postorder(root->right);
    printf("%d ", root->data);
}

int main() {
    struct Node* root = create(10);
    root->left = create(20);
    root->right = create(30);
    root->left->left = create(40);
    root->left->right = create(50);

    printf("Inorder: ");
    inorder(root);
```

```
    printf("\nPreorder: ");
    preorder(root);

    printf("\nPostorder: ");
    postorder(root);

    return 0;
}
```

```
Inorder: 40 20 50 10 30
Preorder: 10 20 40 50 30
Postorder: 40 50 20 30 10

=== Code Execution Successful ===
```

Q14:Write a program to perform insertion and deletion in the Binary Search Tree.

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

struct Node {
    int data;
    struct Node *left, *right;
};

struct Node* create(int x) {
    struct Node* n = malloc(sizeof(struct Node));
    n->data = x;
    n->left = n->right = NULL;
    return n;
}

struct Node* insert(struct Node* root, int x) {
    if (!root) return create(x);
    if (x < root->data)
        root->left = insert(root->left, x);
    else
        root->right = insert(root->right, x);
    return root;
```

```
}

struct Node* minValueNode(struct Node* root) {
    while (root->left)
        root = root->left;
    return root;
}

struct Node* delete(struct Node* root, int x) {
    if (!root) return root;

    if (x < root->data)
        root->left = delete(root->left, x);
    else if (x > root->data)
        root->right = delete(root->right, x);
    else {
        if (!root->left) {
            struct Node* t = root->right;
            free(root);
            return t;
        }
        else if (!root->right) {
            struct Node* t = root->left;
            free(root);
```

```
            return t;
        }
        struct Node* t = minValueNode(root->right);
        root->data = t->data;
        root->right = delete(root->right, t->data);
    }
    return root;
}

void inorder(struct Node* root) {
    if (!root) return;
    inorder(root->left);
    printf("%d ", root->data);
    inorder(root->right);
}

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

    root = insert(root, 50);
    root = insert(root, 30);
    root = insert(root, 70);
    root = insert(root, 20);
    root = insert(root, 40);
```

```
    printf("BST Inorder (After Insertion): ");
    inorder(root);

    root = delete(root, 30);

    printf("\nBST Inorder (After Deletion): ");
    inorder(root);

    return 0;
}
```

```
BST Inorder (After Insertion): 20 30 40 50 70
BST Inorder (After Deletion): 20 40 50 70

=== Code Execution Successful ===
```

Q15:Write a program to represent an undirected graph using the adjacency matrix to implement the graph and perform the operations with menu-driven options for the following tasks:

1. Create graph 2. Insert an edge 3. Print Adjacency Matrix 4. List all vertices that are adjacent to a specified vertex. 6.

Exit program

```
#include <stdio.h>

int graph[20][20];
int n;

void createGraph() {
    int i, j;
    printf("Enter number of vertices: ");
    scanf("%d", &n);

    printf("Enter adjacency matrix (%d x %d):\n", n, n);
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            scanf("%d", &graph[i][j]);
}

void insertEdge() {
```

```
    int u, v;
    printf("Enter edge (u v): ");
    scanf("%d %d", &u, &v);

    graph[u][v] = 1;
    graph[v][u] = 1;
}

void printMatrix() {
    int i, j;
    printf("Adjacency Matrix:\n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++)
            printf("%d ", graph[i][j]);
        printf("\n");
    }
}

void adjacentVertices() {
    int v, i;
    printf("Enter vertex: ");
    scanf("%d", &v);

    printf("Vertices adjacent to %d: ", v);
```

```
    for (i = 0; i < n; i++)
        if (graph[v][i] == 1)
            printf("%d ", i);
}

int main() {
    int ch;

    while (1) {
        printf("\n1.Create Graph\n2.Insert Edge\n3.Print Matrix\n4.Adjacent
Vertices\n5.Exit\n");
        printf("Enter choice: ");
        scanf("%d", &ch);

        switch (ch) {
            case 1: createGraph(); break;
            case 2: insertEdge(); break;
            case 3: printMatrix(); break;
            case 4: adjacentVertices(); break;
            case 5: return 0;
            default: printf("Invalid Choice");
        }
    }
}
```

```
Enter number of vertices: 2
Enter adjacency matrix (2 x 2):
1 2
3 4

1.Create Graph
2.Insert Edge
3.Print Matrix
4.Adjacent Vertices
5.Exit
Enter choice: 4
Enter vertex: 4
Vertices adjacent to 4:
1.Create Graph
2.Insert Edge
3.Print Matrix
4.Adjacent Vertices
5.Exit
Enter choice:
3
Adjacency Matrix:
1 2
3 4
```