# *1 ) To implement traversing, insertion and deletion in arrays.*

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

void traverse(int arr[], int size) {
   printf("Array elements: ");
   for(int i = 0; i < size; i++)
      printf("%d ", arr[i]);
   printf("\n");
}

void insert(int arr[], int *size, int pos, int value) {
   for(int i = *size; i > pos; i--)
      arr[i] = arr[i-1];
   arr[pos] = value;
   (*size)++;
}

void deleteElement(int arr[], int *size, int pos) {
   for(int i = pos; i < *size-1; i++)
      arr[i] = arr[i+1];
   (*size)--;
}

int main() {
   int arr[10] = {1, 2, 3, 4, 5};
   int size = 5;

   clrscr();
   traverse(arr, size);
   insert(arr, &size, 2, 99); // Inserting 99 at position 2
   traverse(arr, size);
   deleteElement(arr, &size, 3); // Deleting element at position 3
   traverse(arr, size);

   getch();
   return 0;
```

}

## *2 ) To implement, addition, Multiplication of Two sparse Matrices.*

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

#define MAX 10

struct Sparse {
   int row, col, val;
};

void inputSparseMatrix(struct Sparse matrix[], int m) {
   printf("Enter the Sparse Matrix elements:\n");
   for(int i = 0; i < m; i++) {
      printf("Row Col Val: ");
      scanf("%d%d%d", &matrix[i].row, &matrix[i].col, &matrix[i].val);
   }
}

void addSparseMatrices(struct Sparse mat1[], int m1, struct Sparse mat2[], int m2) {
   int i = 0, j = 0, k = 0;
   struct Sparse result[MAX];

   while(i < m1 && j < m2) {
      if(mat1[i].row == mat2[j].row && mat1[i].col == mat2[j].col) {
         result[k].row = mat1[i].row;
         result[k].col = mat1[i].col;
         result[k++].val = mat1[i].val + mat2[j].val;
         i++; j++;
      } else if(mat1[i].row < mat2[j].row || (mat1[i].row == mat2[j].row && mat1[i].col <
mat2[j].col)) {
         result[k++] = mat1[i++];
      } else {
         result[k++] = mat2[j++];
      }
```

```
    }

    while(i < m1) result[k++] = mat1[i++];
    while(j < m2) result[k++] = mat2[j++];

    printf("Resultant Sparse Matrix:\n");
    for(int i = 0; i < k; i++) {
        printf("Row: %d Col: %d Value: %d\n", result[i].row, result[i].col, result[i].val);
    }
}

int main() {
    struct Sparse mat1[MAX], mat2[MAX];
    int m1, m2;

    clrscr();
    printf("Enter number of non-zero elements in Matrix 1: ");
    scanf("%d", &m1);
    inputSparseMatrix(mat1, m1);

    printf("Enter number of non-zero elements in Matrix 2: ");
    scanf("%d", &m2);
    inputSparseMatrix(mat2, m2);

    addSparseMatrices(mat1, m1, mat2, m2);

    getch();
    return 0;
}
```

## *3 ) To implement insertion, deletion and pattern matching of a substring in a given string using linked list.*

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

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

void insertNode(struct Node **head, char data, int pos) {
   struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
   newNode->data = data;
   newNode->next = NULL;

   if(pos == 1) {
      newNode->next = *head;
      *head = newNode;
      return;
   }

   struct Node *temp = *head;
   for(int i = 1; i < pos - 1 && temp != NULL; i++) {
      temp = temp->next;
   }
   if(temp != NULL) {
      newNode->next = temp->next;
      temp->next = newNode;
   }
}

void deleteNode(struct Node **head, int pos) {
   if(*head == NULL) return;

   struct Node *temp = *head;
   if(pos == 1) {
      *head = temp->next;
      free(temp);
      return;
   }

   for(int i = 1; temp != NULL && i < pos - 1; i++)
      temp = temp->next;

   if(temp == NULL || temp->next == NULL) return;
```

```
    struct Node *next = temp->next->next;
    free(temp->next);
    temp->next = next;
}

int patternMatch(struct Node *head, char *pattern) {
    struct Node *temp = head;
    int i = 0, found = 0;
    while(temp != NULL) {
        if(temp->data == pattern[i]) i++;
        else i = 0;

        if(i == strlen(pattern)) {
            found = 1;
            break;
        }
        temp = temp->next;
    }
    return found;
}

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

int main() {
    struct Node *head = NULL;
    int choice, pos;
    char value, pattern[100];

    clrscr();

    do {
        printf("\n1. Insert Node\n2. Delete Node\n3. Pattern Matching\n4. Display\n5.
Exit\nEnter your choice: ");
        scanf("%d", &choice);
```

```
        switch(choice) {
            case 1:
                printf("Enter character to insert and position: ");
                scanf(" %c %d", &value, &pos);
                insertNode(&head, value, pos);
                break;
            case 2:
                printf("Enter position to delete: ");
                scanf("%d", &pos);
                deleteNode(&head, pos);
                break;
            case 3:
                printf("Enter pattern to match: ");
                scanf("%s", pattern);
                if(patternMatch(head, pattern))
                    printf("Pattern found!\n");
                else
                    printf("Pattern not found.\n");
                break;
            case 4:
                display(head);
                break;
        }
    } while(choice != 5);

    getch();
    return 0;
}
```

## *4 ) To implement Insertion and deletion in Singly Linked List at Given Location as well as for a Given Item in sorted List.*

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

struct Node {
```

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

void insertAtPosition(struct Node** head, int data, int pos) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;

    if(pos == 1) {
        newNode->next = *head;
        *head = newNode;
        return;
    }

    struct Node* temp = *head;
    for(int i = 1; i < pos - 1 && temp != NULL; i++) {
        temp = temp->next;
    }
    if(temp != NULL) {
        newNode->next = temp->next;
        temp->next = newNode;
    }
}

void deleteAtPosition(struct Node** head, int pos) {
    if(*head == NULL) return;

    struct Node* temp = *head;
    if(pos == 1) {
        *head = temp->next;
        free(temp);
        return;
    }

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

    if(temp == NULL || temp->next == NULL) return;
```

```
    struct Node* next = temp->next->next;
    free(temp->next);
    temp->next = next;
}

void insertSorted(struct Node** head, int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;

    if(*head == NULL || (*head)->data >= data) {
        newNode->next = *head;
        *head = newNode;
        return;
    }

    struct Node* temp = *head;
    while(temp->next != NULL && temp->next->data < data) {
        temp = temp->next;
    }
    newNode->next = temp->next;
    temp->next = newNode;
}

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

int main() {
    struct Node* head = NULL;
    clrscr();

    insertAtPosition(&head, 10, 1);
    insertAtPosition(&head, 20, 2);
    insertAtPosition(&head, 5, 1); // Inserting at given location
    display(head);
```

```
    insertSorted(&head, 15); // Inserting in sorted order
    display(head);

    deleteAtPosition(&head, 2); // Deleting at a given location
    display(head);

    getch();
    return 0;
}
```

## *5 ) To Implement Insertion and deletion in Circular Linked List.*

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

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

void insertEnd(struct Node **head, int data) {
    struct Node *newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = *head;

    if(*head == NULL) {
        *head = newNode;
        newNode->next = *head;
        return;
    }

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

```
}

void deleteNode(struct Node **head, int pos) {
   if(*head == NULL) return;

   struct Node *temp = *head, *prev;
   if(pos == 1) {
      while(temp->next != *head) {
         temp = temp->next;
      }
      temp->next = (*head)->next;
      free(*head);
      *head = temp->next;
      return;
   }

   int count = 1;
   while(count < pos - 1) {
      temp = temp->next;
      count++;
   }
   prev = temp->next;
   temp->next = prev->next;
   free(prev);
}

void display(struct Node *head) {
   if(head == NULL) return;

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

int main() {
   struct Node *head = NULL;
   clrscr();
```

```
    insertEnd(&head, 10);
    insertEnd(&head, 20);
    insertEnd(&head, 30);
    display(head);

    deleteNode(&head, 2); // Deleting at position 2
    display(head);

    getch();
    return 0;
}
```

## *6 ) To implement insertion and Deletion in Stack and Queue using arrays and pointer.*

→ **STACK**

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

#define MAX 5
int stack[MAX];
int top = -1;

void push(int value) {
    if(top == MAX-1) {
        printf("Stack Overflow!\n");
        return;
    }
    stack[++top] = value;
}

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

```
}

void displayStack() {
   if(top == -1) {
      printf("Stack is empty!\n");
      return;
   }
   printf("Stack elements: ");
   for(int i = top; i >= 0; i--)
      printf("%d ", stack[i]);
   printf("\n");
}

int main() {
   clrscr();
   push(10);
   push(20);
   push(30);
   displayStack();
   pop();
   displayStack();
   getch();
   return 0;
}
```

→ **QUEUE**

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

#define MAX 5
int queue[MAX];
int front = -1, rear = -1;

void enqueue(int value) {
   if(rear == MAX-1) {
      printf("Queue Overflow!\n");
      return;
   }
   if(front == -1) front = 0;
```

```
    queue[++rear] = value;
}

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

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

int main() {
    clrscr();
    enqueue(10);
    enqueue(20);
    enqueue(30);
    displayQueue();
    dequeue();
    displayQueue();
    getch();
    return 0;
}
```

## *7 ) To implement Fibonacci Series and Tower of Hanoi Using Recursion.*

→ **Fibonacci Series**

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

int fibonacci(int n) {
   if(n <= 1)
      return n;
   return fibonacci(n-1) + fibonacci(n-2);
}

int main() {
   int n;
   clrscr();
   printf("Enter the number of terms: ");
   scanf("%d", &n);
   printf("Fibonacci Series: ");
   for(int i = 0; i < n; i++) {
      printf("%d ", fibonacci(i));
   }
   getch();
   return 0;
}
```

→ **Tower of Hanoi**

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

void hanoi(int n, char from, char to, char aux) {
   if(n == 1) {
      printf("Move disk 1 from %c to %c\n", from, to);
      return;
   }
   hanoi(n-1, from, aux, to);
   printf("Move disk %d from %c to %c\n", n, from, to);
   hanoi(n-1, aux, to, from);
}

int main() {
   int n;
```

```
    clrscr();
    printf("Enter number of disks: ");
    scanf("%d", &n);
    hanoi(n, 'A', 'C', 'B');
    getch();
    return 0;
}
```

# *8 ) Creation of Trees and Tree Traversal Algorithms: Recursive and Non-Recursive.*

→ **Binary Tree with Recursive Traversals**

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

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

struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->left = newNode->right = NULL;
    return newNode;
}

// Recursive traversals
void inorder(struct Node* root) {
    if(root == NULL) return;
    inorder(root->left);
    printf("%d ", root->data);
    inorder(root->right);
}

void preorder(struct Node* root) {
```

```
    if(root == NULL) return;
    printf("%d ", root->data);
    preorder(root->left);
    preorder(root->right);
}

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

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

    printf("Inorder Traversal: ");
    inorder(root);
    printf("\nPreorder Traversal: ");
    preorder(root);
    printf("\nPostorder Traversal: ");
    postorder(root);

    getch();
    return 0;
}
```

→ **Non-Recursive Inorder Traversal**

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

struct Node {
    int data;
```

```
    struct Node *left, *right;
};

struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->left = newNode->right = NULL;
    return newNode;
}

void inorderNonRecursive(struct Node* root) {
    struct Node *stack[100];
    int top = -1;
    struct Node* curr = root;

    while(curr != NULL || top != -1) {
        while(curr != NULL) {
            stack[++top] = curr;
            curr = curr->left;
        }
        curr = stack[top--];
        printf("%d ", curr->data);
        curr = curr->right;
    }
}

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

    printf("Non-Recursive Inorder Traversal: ");
    inorderNonRecursive(root);

    getch();
    return 0;
}
```

# *9 ) Creation of Graphs and Graph Traversal Algorithms. Sorting: Insertion Sort Quick Sort Merge Sort Bubble Sort Heap Sort*

→ **Adjacency Matrix for Graph**

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

#define MAX 5
int graph[MAX][MAX];

void createGraph() {
   int i, j, edges, origin, dest;
   printf("Enter the number of edges: ");
   scanf("%d", &edges);

   for(i = 0; i < edges; i++) {
      printf("Enter edge (origin destination): ");
      scanf("%d %d", &origin, &dest);
      graph[origin][dest] = 1;
      graph[dest][origin] = 1; // For undirected graph
   }
}

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

int main() {
```

```
    clrscr();
    createGraph();
    displayGraph();
    getch();
    return 0;
}
```

### → BFS and DFS for Graph

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

#define MAX 5
int graph[MAX][MAX], visited[MAX];

void bfs(int start) {
    int queue[MAX], front = 0, rear = 0, i;
    queue[rear] = start;
    visited[start] = 1;
    printf("BFS: %d ", start);

    while(front <= rear) {
        start = queue[front++];
        for(i = 0; i < MAX; i++) {
            if(graph[start][i] == 1 && visited[i] == 0) {
                queue[++rear] = i;
                visited[i] = 1;
                printf("%d ", i);
            }
        }
    }
}

void dfs(int start) {
    int i;
    visited[start] = 1;
    printf("%d ", start);

    for(i = 0; i < MAX; i++) {
        if(graph[start][i] == 1 && visited[i] == 0) {
```

```
            dfs(i);
        }
    }
}

int main() {
    clrscr();
    createGraph();  // Use the previous graph creation function
    printf("\nDFS Traversal: ");
    for(int i = 0; i < MAX; i++) {
        visited[i] = 0;  // Reset visited for DFS
    }
    dfs(0);  // Start DFS from node 0
    getch();
    return 0;
}
```

# *10 ) Implementation of Sparse Matrix and Polynomial using Link list.*

→ **Sparse Matrix using Linked List**

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

struct Node {
    int row, col, val;
    struct Node* next;
};

struct Node* createNode(int row, int col, int val) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->row = row;
    newNode->col = col;
    newNode->val = val;
    newNode->next = NULL;
    return newNode;
```

```
}

void insertNode(struct Node** head, int row, int col, int val) {
   struct Node* newNode = createNode(row, col, val);
   newNode->next = *head;
   *head = newNode;
}

void displaySparseMatrix(struct Node* head) {
   struct Node* temp = head;
   printf("Row Col Value\n");
   while(temp != NULL) {
      printf("%d    %d    %d\n", temp->row, temp->col, temp->val);
      temp = temp->next;
   }
}

int main() {
   struct Node* head = NULL;
   clrscr();

   insertNode(&head, 0, 0, 3);
   insertNode(&head, 0, 2, 5);
   insertNode(&head, 1, 1, 8);
   insertNode(&head, 2, 2, 9);

   displaySparseMatrix(head);

   getch();
   return 0;
}
```

→ **Polynomial Representation using Linked List**

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

struct PolyNode {
   int coeff;
   int power;
```

```
    struct PolyNode* next;
};

struct PolyNode* createPolyNode(int coeff, int power) {
    struct PolyNode* newNode = (struct PolyNode*)malloc(sizeof(struct PolyNode));
    newNode->coeff = coeff;
    newNode->power = power;
    newNode->next = NULL;
    return newNode;
}

void insertPolyNode(struct PolyNode** head, int coeff, int power) {
    struct PolyNode* newNode = createPolyNode(coeff, power);
    newNode->next = *head;
    *head = newNode;
}

void displayPoly(struct PolyNode* head) {
    struct PolyNode* temp = head;
    while(temp != NULL) {
        printf("%dx^%d", temp->coeff, temp->power);
        temp = temp->next;
        if(temp != NULL) printf(" + ");
    }
    printf("\n");
}

int main() {
    struct PolyNode* poly1 = NULL;
    clrscr();

    insertPolyNode(&poly1, 3, 2); // 3x^2
    insertPolyNode(&poly1, 5, 1); // 5x^1
    insertPolyNode(&poly1, 6, 0); // 6x^0

    printf("Polynomial 1: ");
    displayPoly(poly1);

    getch();
    return 0;
}
```