```
%%writefile prog1.c
//1.Program to Initialzation of malloc() function into 0
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int n = 5;
    int *arr;
    arr = (int *)malloc(n * sizeof(int));
    for(int i = 0; i < n; i++)
    {
        arr[i] = 0;
    }
    for(int i = 0; i < n; i++)
    {
        printf("%d ", arr[i]);
    }
    free(arr);
    printf("\nMAC Address:24:6A:0E:DD:3A:0F\n");
    return 0;
}
```

```
Writing prog1.c
```

```
%%bash
gcc prog1.c -o prog1
./prog1
```

```
0 0 0 0 0
MAC Address:24:6A:0E:DD:3A:0F
```

```
%%writefile prog2.c
//2.Program to print the address of the pointer
#include <stdio.h>
int main()
{
    int a = 10;
    unsigned long addr = (unsigned long)&a;
    int count = 0;
    unsigned long temp = addr;
    while(temp > 0)
    {
        count++;
        temp /= 10;
    }
    printf("Address: %lu\n", addr);
    printf("Digits: %d\n", count);
    printf("\nMAC Address:24:6A:0E:DD:3A:0F\n");
    return 0;
}
```

```
Writing prog2.c
```

```
%%bash
gcc prog2.c -o prog2
./prog2
```

```
Address: 140731377718624
Digits: 15

MAC Address:24:6A:0E:DD:3A:0F
```

```
%%writefile selection_sort.c
//3.Program to sort an array using selection sort with dynamic memory allocation
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int n, i, j, min, temp;
    int *arr;
    printf("Enter number of elements: ");
    scanf("%d", &n);
    arr = (int *)malloc(n * sizeof(int));
    printf("Enter elements:\n");
```

```
    for(i = 0; i < n; i++)
    {
        scanf("%d", &arr[i]);
    }
    for(i = 0; i < n - 1; i++)
    {
        min = i;
        for(j = i + 1; j < n; j++)
        {
            if(arr[j] < arr[min])
             {
                min = j;
            }
        }
        temp = arr[i];
        arr[i] = arr[min];
        arr[min] = temp;
    }
    printf("Sorted array:\n");
    for(i = 0; i < n; i++)
    {
        printf("%d ", arr[i]);
    }
    free(arr);
    printf("\nMAC Address:24:6A:0E:DD:3A:0F\n");
    return 0;
}
```

```
Writing selection_sort.c
```

```
%%writefile input.txt
5
64 25 12 22 11
```

```
Writing input.txt
```

```
%%bash
gcc selection_sort.c -o selection_sort
./selection_sort < input.txt
```

```
Enter number of elements: Enter elements:
Sorted array:
11 12 22 25 64
MAC Address:24:6A:0E:DD:3A:0F
```

```
%%writefile selection_static.c
//4.Program to sort an array using selection sort without dynamic memory allocation
#include <stdio.h>
int main()
 {
    int n, i, j, min, temp;
    int arr[50];
    printf("Enter number of elements: ");
    scanf("%d", &n);
    printf("Enter elements:\n");
    for(i = 0; i < n; i++)
     {
        scanf("%d", &arr[i]);
    }
    for(i = 0; i < n - 1; i++)
    {
        min = i;
        for(j = i + 1; j < n; j++)
        {
            if(arr[j] < arr[min])
             {
                min = j;
            }
        }
        temp = arr[i];
        arr[i] = arr[min];
        arr[min] = temp;
    }
    printf("Sorted array:\n");
    for(i = 0; i < n; i++)
     {
```

```
            printf("%d ", arr[i]);
        }
        printf("\nMAC Address:24:6A:0E:DD:3A:0F\n");
        return 0;
    }
```

```
Writing selection_static.c
```

```
%%bash
gcc selection_static.c -o selection_static
printf "5\n64 25 12 22 11\n" | ./selection_static
```

```
Enter number of elements: Enter elements:
Sorted array:
11 12 22 25 64
MAC Address:24:6A:0E:DD:3A:0F
```

```
%%writefile two_d_array.c
//5.Program to declare the two dimensional array using dynamic memory allocation
#include <stdio.h>
#include <stdlib.h>
int main()
 {
    int rows, cols;
    int **arr;
    printf("Enter number of rows: ");
    scanf("%d", &rows);
    printf("Enter number of columns: ");
    scanf("%d", &cols);
    arr = (int **)malloc(rows * sizeof(int *));
    for(int i = 0; i < rows; i++)
    {
        arr[i] = (int *)malloc(cols * sizeof(int));
    }
    printf("2-D array declared successfully\n");
    for(int i = 0; i < rows; i++)
    {
        free(arr[i]);
    }
    free(arr);
    printf("\nMAC Address:24:6A:0E:DD:3A:0F\n");
    return 0;
}
```

```
Writing two_d_array.c
```

```
%%bash
gcc two_d_array.c -o two_d_array
printf "3\n4\n" | ./two_d_array
```

```
Enter number of rows: Enter number of columns: 2-D array declared successfully

MAC Address:24:6A:0E:DD:3A:0F
```

```
%%writefile pattern_match.c
//6.Program to match the pattern of the strings
#include <stdio.h>
#include <string.h>
int main()
{
    char text[50], pattern[50];
    int i, j, found;
    printf("Enter text: ");
    scanf("%s", text);
    printf("Enter pattern: ");
    scanf("%s", pattern);
    int n = strlen(text);
    int m = strlen(pattern);
    for(i = 0; i <= n - m; i++)
     {
        found = 1;
        for(j = 0; j < m; j++)
        {
            if(text[i + j] != pattern[j])
            {
                found = 0;
```

```
                break;
            }
        }
        if(found)
        {
            printf("\nPattern found at position %d\n", i);
        }
    }
    printf("\nMAC Address:24:6A:0E:DD:3A:0F\n");
    return 0;
}
```

```
Writing pattern_match.c
```

```
%%bash
gcc pattern_match.c -o pattern_match
printf "ABCDABCD\nABC\n" | ./pattern_match
```

```
Enter text: Enter pattern:
Pattern found at position 0

Pattern found at position 4

MAC Address:24:6A:0E:DD:3A:0F
```

```
%%writefile self_ref_simple.c
//7.Program to create a self referencial structure
#include <stdio.h>
#include <stdlib.h>
struct sample {
    int value;
    struct sample *ref;
};
int main()
{
    struct sample a, b;
    a.value = 100;
    a.ref = &b;
    b.value = 200;
    b.ref = NULL;
    printf("Value of a: %d\n", a.value);
    printf("Value of b using a.ref: %d\n", a.ref->value);
    printf("\nMAC Address:24:6A:0E:DD:3A:0F\n");
    return 0;
}
```

```
Writing self_ref_simple.c
```

```
%%bash
gcc self_ref_simple.c -o self_ref_simple
./self_ref_simple
```

```
Value of a: 100
Value of b using a.ref: 200

MAC Address:24:6A:0E:DD:3A:0F
```

```
%%writefile increment.c
//8.Program to perfrom post & pre increment operation
#include <stdio.h>
int main()
{
    int a = 5, b = 5;
    printf("Pre-increment (++a): %d\n", ++a);
    printf("Value of a after pre-increment: %d\n", a);
    printf("Post-increment (b++): %d\n", b++);
    printf("Value of b after post-increment: %d\n", b);
    printf("\nMAC Address:24:6A:0E:DD:3A:0F\n");
    return 0;
}
```

```
Writing increment.c
```

```
%%bash
gcc increment.c -o increment
./increment
```

```
Pre-increment (++a): 6
Value of a after pre-increment: 6
Post-increment (b++): 5
Value of b after post-increment: 6

MAC Address:24:6A:0E:DD:3A:0F
```

```
%%writefile decrement.c
//9.Program to perfrom post & pre decrement operation
#include <stdio.h>
int main()
 {
    int a = 5, b = 5;
    printf("Pre-decrement (--a): %d\n", --a);
    printf("Value of a after pre-decrement: %d\n", a);
    printf("Post-decrement (b--): %d\n", b--);
    printf("Value of b after post-decrement: %d\n", b);
    printf("\nMAC Address:24:6A:0E:DD:3A:0F\n");
    return 0;
}
```

```
Writing decrement.c
```

```
%%bash
gcc decrement.c -o decrement
./decrement
```

```
Pre-decrement (--a): 4
Value of a after pre-decrement: 4
Post-decrement (b--): 5
Value of b after post-decrement: 4

MAC Address:24:6A:0E:DD:3A:0F
```

```
%%writefile regular_queue.c
//10.Program to create regular queue
#include <stdio.h>
#define SIZE 5
int queue[SIZE];
int front = -1, rear = -1;
void enqueue(int x)
{
    if (rear == SIZE - 1)
     {
        printf("Queue Overflow\n");
        return;
    }
    if (front == -1)
        front = 0;
    queue[++rear] = x;
    printf("Inserted %d\n", x);
}
void dequeue()
 {
    if (front == -1 || front > rear)
     {
        printf("Queue Underflow\n");
        return;
    }
    printf("Deleted %d\n", queue[front++]);
}
void display()
{
    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()
{
    enqueue(10);
    enqueue(20);
    enqueue(30);
    display();
    dequeue();
    display();
    printf("\nMAC Address:24:6A:0E:DD:3A:0F\n");
    return 0;
}
```

```
Writing regular_queue.c
```

```
%%bash
gcc regular_queue.c -o regular_queue
./regular_queue
```

```
Inserted 10
Inserted 20
Inserted 30
Queue elements: 10 20 30
Deleted 10
Queue elements: 20 30

MAC Address:24:6A:0E:DD:3A:0F
```

```
%%writefile linked_list.c
//11. Program to perfrom operations of linked lists(traversal , insertion, deletion)
#include <stdio.h>
#include <stdlib.h>
struct node
 {
    int data;
    struct node *next;
};
struct node *head = NULL;
void traverse()
 {
    struct node *temp = head;
    if (temp == NULL)
    {
        printf("List is empty\n");
        return;
    }
    printf("Linked List: ");
    while (temp != NULL)
    {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}
void insert(int value)
 {
    struct node *newNode = (struct node *)malloc(sizeof(struct node));
    newNode->data = value;
    newNode->next = head;
    head = newNode;
}
void delete()
{
    if (head == NULL)
    {
        printf("List is empty\n");
        return;
    }
    struct node *temp = head;
    head = head->next;
    printf("Deleted element: %d\n", temp->data);
    free(temp);
}
int main()
{
    insert(10);
    insert(20);
    insert(30);
```

```
    traverse();
    delete();
    traverse();
    printf("\nMAC Address:24:6A:0E:DD:3A:0F\n");
    return 0;
}
```

```
Writing linked_list.c
```

```
%%bash
gcc linked_list.c -o linked_list
./linked_list
```

```
Linked List: 30 20 10
Deleted element: 30
Linked List: 20 10

MAC Address:24:6A:0E:DD:3A:0F
```

```
%%writefile stack_to_list.c
//12.Program to perfrom linked list using stack
#include <stdio.h>
#include <stdlib.h>
struct stack
{
    int data;
    struct stack *next;
};
struct node
{
    int data;
    struct node *next;
};
struct stack *top = NULL;
struct node *head = NULL;
void push(int x)
 {
    struct stack *temp = (struct stack *)malloc(sizeof(struct stack));
    temp->data = x;
    temp->next = top;
    top = temp;
}
void createList()
{
    struct node *newNode;
    while (top != NULL) {
        newNode = (struct node *)malloc(sizeof(struct node));
        newNode->data = top->data;
        newNode->next = head;
        head = newNode;
        struct stack *temp = top;
        top = top->next;
        free(temp);
    }
}
void display()
{
    struct node *temp = head;
    printf("Linked List: ");
    while (temp != NULL)
     {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}
int main()
{
    push(10);
    push(20);
    push(30);
    createList();
    display();
    printf("\nMAC Address:24:6A:0E:DD:3A:0F\n");
    return 0;
}
```

```
Writing stack_to_list.c
```

```
%%bash
gcc stack_to_list.c -o stack_to_list
./stack_to_list
```

```
Linked List: 10 20 30

MAC Address:24:6A:0E:DD:3A:0F
```

```
%%writefile queue_of_list.c
//13.Program to create linked list using queue
#include <stdio.h>
#include <stdlib.h>
struct node
{
    int data;
    struct node *next;
};
struct node *front = NULL;
struct node *rear = NULL;
void enqueue(int value)
{
    struct node *newNode = (struct node *)malloc(sizeof(struct node));
    newNode->data = value;
    newNode->next = NULL;
    if (rear == NULL)
   {
        front = rear = newNode;
    }
    else
    {
        rear->next = newNode;
        rear = newNode;
    }
}
void dequeue()
 {
    if (front == NULL)
     {
        printf("Queue is empty\n");
        return;
    }
    struct node *temp = front;
    front = front->next;
    if (front == NULL)
        rear = NULL;
        free(temp);
}
void display()
 {
    struct node *temp = front;
    if (temp == NULL)
    {
        printf("Queue is empty\n");
        return;
    }
    printf("Queue elements: ");
    while (temp != NULL)
    {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}
int main()
 {
    enqueue(10);
    enqueue(20);
    enqueue(30);
    display();
    dequeue();
    display();
    printf("\nMAC Address:24:6A:0E:DD:3A:0F\n");
    return 0;
}
```

```
Writing queue_of_list.c
```

```
%%bash
gcc queue_of_list.c -o queue_of_list
./queue_of_list
```

```
Queue elements: 10 20 30
Queue elements: 20 30

MAC Address:24:6A:0E:DD:3A:0F
```

```
%%writefile time.c
// 14.Program to print time taken for arrays and linked lists during insertion and deletion
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
struct Node
{
    int data;
    struct Node* next;
};
void insertLinkedList(struct Node** head, int value)
{
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = *head;
    *head = newNode;
}
void deleteLinkedList(struct Node** head)
 {
    if (*head == NULL) return;
    struct Node* temp = *head;
    *head = (*head)->next;
    free(temp);
}
int main()
{
    int n = 100000;
    int *arr = (int*)malloc(n * sizeof(int));
    struct Node* head = NULL;
    clock_t start, end;
    double time_taken;
    start = clock();
    for (int i = 0; i < n; i++)
     {
        for (int j = i; j > 0; j--)
            arr[j] = arr[j - 1];
        arr[0] = i;
    }
    end = clock();
    time_taken = ((double)(end - start)) / CLOCKS_PER_SEC;
    printf("Array insertion at beginning: %f seconds\n", time_taken);
    start = clock();
    for (int i = 0; i < n; i++)
     {
        for (int j = 0; j < n - i - 1; j++)
            arr[j] = arr[j + 1];
    }
    end = clock();
    time_taken = ((double)(end - start)) / CLOCKS_PER_SEC;
    printf("Array deletion from beginning: %f seconds\n", time_taken);
    start = clock();
    for (int i = 0; i < n; i++)
        insertLinkedList(&head, i);
    end = clock();
    time_taken = ((double)(end - start)) / CLOCKS_PER_SEC;
    printf("Linked list insertion at beginning: %f seconds\n", time_taken);
    start = clock();
    for (int i = 0; i < n; i++)
        deleteLinkedList(&head);
    end = clock();
    time_taken = ((double)(end - start)) / CLOCKS_PER_SEC;
    printf("Linked list deletion from beginning: %f seconds\n", time_taken);
    free(arr);
    printf("\nMAC Address:24:6A:0E:DD:3A:0F\n");
    return 0;
}
```

```
Writing time.c
```

```
%%bash
gcc time.c -o time
./time
```

```
Array insertion at beginning: 14.842677 seconds
Array deletion from beginning: 15.709465 seconds
Linked list insertion at beginning: 0.004066 seconds
```

```
Linked list deletion from beginning: 0.001701 seconds

MAC Address:24:6A:0E:DD:3A:0F
```

```
%%writefile sparse.c
//15.Program to create sparse matrix
#include <stdio.h>
int main()
{
    int a[10][10], i, j, r, c;
    int count = 0;
    printf("Enter rows and columns: ");
    scanf("%d %d", &r, &c);
    printf("Enter matrix elements:\n");
    for(i = 0; i < r; i++)
     {
        for(j = 0; j < c; j++)
        {
            scanf("%d", &a[i][j]);
            if(a[i][j] != 0)
                count++;
        }
    }
    printf("\nSparse Matrix Representation (Row Col Value):\n");
    printf("%d %d %d\n", r, c, count);
    for(i = 0; i < r; i++)
     {
        for(j = 0; j < c; j++)
         {
            if(a[i][j] != 0)
                printf("%d %d %d\n", i, j, a[i][j]);
        }
    }
    printf("\nMAC Address:24:6A:0E:DD:3A:0F\n");
    return 0;
}
```

```
%%bash
gcc sparse.c -o sparse
printf "3 3\n0 0 3\n0 4 0\n0 0 5\n" | ./sparse
```

```
%%writefile polynomial.c
//16.Program to create polynomial expression
#include <stdio.h>
int main()
{
    int n, i;
    int coef[10], power[10];
    printf("\nEnter number of terms:\n");
    scanf("%d", &n);
    for(i = 0; i < n; i++)
    {
        printf("\nEnter coefficient and power:\n ");
        scanf("%d %d", &coef[i], &power[i]);
    }
    printf("\nPolynomial Expression:\n");
    for(i = 0; i < n; i++)
     {
        printf("%dx^%d", coef[i], power[i]);
        if(i != n - 1)
            printf(" + ");
    }
    printf("\nMAC Address:24:6A:0E:DD:3A:0F\n");
    return 0;
}
```

```
Writing polynomial.c
```

```
%%bash
gcc polynomial.c -o polynomial
printf "3\n5 2\n3 1\n1 0\n" | ./polynomial
```

```
Enter number of terms:
```

```
Enter coefficient and power:

Enter coefficient and power:

Enter coefficient and power:

Polynomial Expression:
5x^2 + 3x^1 + 1x^0
MAC Address:24:6A:0E:DD:3A:0F
```

```
%%writefile tree.c
//17.Program to create a tree
#include <stdio.h>
#include <stdlib.h>
struct node
{
    int data;
    struct node *left;
    struct node *right;
};
int main()
{
    struct node *root, *leftChild, *rightChild;
    root = (struct node *)malloc(sizeof(struct node));
    root->data = 10;
    leftChild = (struct node *)malloc(sizeof(struct node));
    leftChild->data = 20;
    rightChild = (struct node *)malloc(sizeof(struct node));
    rightChild->data = 30;
    root->left = leftChild;
    root->right = rightChild;
    leftChild->left = NULL;
    leftChild->right = NULL;
    rightChild->left = NULL;
    rightChild->right = NULL;
    printf("Root: %d\n", root->data);
    printf("Left Child: %d\n", root->left->data);
    printf("Right Child: %d\n", root->right->data);
    printf("\nMAC Address:24:6A:0E:DD:3A:0F\n");
    return 0;
}
```

```
Writing tree.c
```

```
%%bash
gcc tree.c -o tree
./tree
```

```
Root: 10
Left Child: 20
Right Child: 30

MAC Address:24:6A:0E:DD:3A:0F
```

```
%%writefile array_tree.c
//18.Program to using array construct a tree
#include <stdio.h>
int main()
{
    int tree[20];
    int n;
    printf("Enter number of nodes: ");
    scanf("%d", &n);
    printf("Enter elements of tree:\n");
    for(int i = 1; i <= n; i++)
     {
        scanf("%d", &tree[i]);
    }
    printf("\nTree representation using array:\n");
    for(int i = 1; i <= n; i++)
    {
        printf("Node at index %d = %d\n", i, tree[i]);
    }
    printf("\nMAC Address:24:6A:0E:DD:3A:0F\n");
    return 0;
}
```

```
Writing array_tree.c
```

```
%%bash
gcc array_tree.c -o array_tree
printf "5\n10 20 30 40 50\n" | ./array_tree
```

```
Enter number of nodes: Enter elements of tree:

Tree representation using array:
Node at index 1 = 10
Node at index 2 = 20
Node at index 3 = 30
Node at index 4 = 40
Node at index 5 = 50

MAC Address:24:6A:0E:DD:3A:0F
```

```
%%writefile tree_queue.c
//19.Program to construct a binary tree using queues
#include <stdio.h>
#include <stdlib.h>
struct node
 {
    int data;
    struct node *left;
    struct node *right;
};
struct queue {
    struct node *data[20];
    int front, rear;
};
struct queue* createQueue()
{
    struct queue *q = (struct queue*)malloc(sizeof(struct queue));
    q->front = q->rear = 0;
    return q;
}
void enqueue(struct queue *q, struct node *temp)
 {
    q->data[q->rear++] = temp;
}
struct node* dequeue(struct queue *q)
 {
    return q->data[q->front++];
}
struct node* newNode(int data)
 {
    struct node *temp = (struct node*)malloc(sizeof(struct node));
    temp->data = data;
    temp->left = temp->right = NULL;
    return temp;
}
void inorder(struct node *root)
 {
    if (root == NULL)
        return;
    inorder(root->left);
    printf("%d ", root->data);
    inorder(root->right);
}

int main()
{
    int n, value;
    struct node *root = NULL;
    printf("Enter number of nodes: ");
    scanf("%d", &n);
    struct queue *q = createQueue();
    printf("Enter node values:\n");
    for (int i = 0; i < n; i++)
    {
        scanf("%d", &value);
        struct node *temp = newNode(value);
       if (root == NULL)
        {
            root = temp;
```

```
            enqueue(q, root);
        }
        else
         {
            struct node *parent = q->data[q->front];
            if (parent->left == NULL)
             {
                parent->left = temp;
                enqueue(q, temp);
            }
            else if (parent->right == NULL)
             {
                parent->right = temp;
                enqueue(q, temp);
                q->front++;
            }
        }
    }
    printf("Inorder traversal of tree:\n");
    inorder(root);
    printf("\nMAC Address:24:6A:0E:DD:3A:0F\n");
    return 0;
}
```

```
Overwriting tree_queue.c
```

```
%%bash
gcc tree_queue.c -o tree_queue
printf "7\n10 20 30 40 50 60 70\n" | ./tree_queue
```

```
Enter number of nodes: Enter node values:
Inorder traversal of tree:
40 20 50 10 60 30 70
MAC Address:24:6A:0E:DD:3A:0F
```

```
%%writefile insert_tree.c
//20.Program to insert a node to the tree
#include <stdio.h>
#include <stdlib.h>
struct node
{
    int data;
    struct node *next;
};
struct node *head = NULL;
void insertBegin(int value)
 {
    struct node *newNode = (struct node*)malloc(sizeof(struct node));
    newNode->data = value;
    newNode->next = head;
    head = newNode;
}
void insertEnd(int value)
{
    struct node *newNode = (struct node*)malloc(sizeof(struct node));
    newNode->data = value;
    newNode->next = NULL;
    if (head == NULL)
    {
        head = newNode;
        return;
    }
    struct node *temp = head;
    while (temp->next != NULL)
        temp = temp->next;
        temp->next = newNode;
}
void display()
 {
    struct node *temp = head;
    printf("Linked List: ");
    while (temp != NULL)
    {
        printf("%d ", temp->data);
        temp = temp->next;
    }
```

```
    printf("\n");
}
int main()
 {
    insertBegin(10);
    insertBegin(20);
    insertEnd(30);
    insertEnd(40);
    display();
    printf("\nMAC Address:24:6A:0E:DD:3A:0F\n");
    return 0;
}
```

```
Overwriting insert_tree.c
```

```
%%bash
gcc insert_tree.c -o insert_tree
./insert_tree
```

```
Linked List: 20 10 30 40

MAC Address:24:6A:0E:DD:3A:0F
```

```
%%writefile delete_tree.c
//21.Program to delete a node from the tree
#include <stdio.h>
#include <stdlib.h>
struct node
{
    int data;
    struct node *next;
};
struct node *head = NULL;
void insertEnd(int value)
 {
    struct node *newNode = (struct node*)malloc(sizeof(struct node));
    newNode->data = value;
    newNode->next = NULL;
    if (head == NULL)
     {
        head = newNode;
        return;
    }
    struct node *temp = head;
    while (temp->next != NULL)
        temp = temp->next;
    temp->next = newNode;
}
void deleteBegin()
 {
    if (head == NULL)
     {
        printf("List is empty\n");
        return;
    }
    struct node *temp = head;
    head = head->next;
    printf("Deleted: %d\n", temp->data);
    free(temp);
}
void deleteEnd()
 {
    if (head == NULL)
     {
        printf("List is empty\n");
        return;
    }
    if (head->next == NULL)
    {
        printf("Deleted: %d\n", head->data);
        free(head);
        head = NULL;
        return;
    }
    struct node *temp = head;
    while (temp->next->next != NULL)
```

```
        temp = temp->next;
    printf("Deleted: %d\n", temp->next->data);
    free(temp->next);
    temp->next = NULL;
}
void deleteAtPos(int pos)
 {
    if (head == NULL || pos <= 0)
    {
        printf("Invalid operation\n");
        return;
    }
    if (pos == 1)
    {
        deleteBegin();
        return;
    }
    struct node *temp = head;
    for (int i = 1; i < pos - 1 && temp->next != NULL; i++)
        temp = temp->next;
   if (temp->next == NULL)
    {
        printf("Position out of range\n");
        return;
    }
    struct node *del = temp->next;
    temp->next = del->next;
    printf("Deleted: %d\n", del->data);
    free(del);
}
void display()
 {
    struct node *temp = head;
    printf("Linked List: ");
    while (temp != NULL)
     {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}
int main()
 {
    insertEnd(10);
    insertEnd(20);
    insertEnd(30);
    insertEnd(40);
    display();
    deleteBegin();
    display();
    deleteEnd();
    display();
    deleteAtPos(2);
    display();
    printf("\nMAC Address:24:6A:0E:DD:3A:0F\n");
    return 0;
}
```

```
Overwriting delete_tree.c
```

```
%%bash
gcc delete_tree.c -o delete_tree
./delete_tree
```

```
Linked List: 10 20 30 40
Deleted: 10
Linked List: 20 30 40
Deleted: 40
Linked List: 20 30
Deleted: 30
Linked List: 20

MAC Address:24:6A:0E:DD:3A:0F
```

```
%%writefile dfs.c
//22.Program to construct depth first search graph
#include <stdio.h>
```

```
void dfs(int adj[10][10], int visited[10], int v, int n)
{
    printf("%d ", v);
    visited[v] = 1;
    for (int i = 0; i < n; i++)
    {
        if (adj[v][i] == 1 && visited[i] == 0)
        {
            dfs(adj, visited, i, n);
        }
    }
}
int main()
{
    int n;
    int adj[10][10], visited[10] = {0};
    printf("Enter number of vertices: ");
    scanf("%d", &n);
    printf("Enter adjacency matrix:\n");
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            scanf("%d", &adj[i][j]);
    printf("DFS traversal starting from vertex 0:\n");
    dfs(adj, visited, 0, n);
    printf("\nMAC Address:24:6A:0E:DD:3A:0F\n");
    return 0;
}
```

```
Overwriting dfs.c
```

```
%%bash
gcc dfs.c -o dfs
printf "4\n0 1 1 0\n1 0 1 0\n1 1 0 1\n0 0 1 0\n" | ./dfs
```

```
Enter number of vertices: Enter adjacency matrix:
DFS traversal starting from vertex 0:
0 1 2 3
MAC Address:24:6A:0E:DD:3A:0F
```

```
%%writefile bfs.c
//23.Program to construct breadth first search graph
#include <stdio.h>
#include <stdlib.h>
struct node
 {
    int data;
    struct node *left;
    struct node *right;
};
struct node* newNode(int data)
{
    struct node* temp = (struct node*)malloc(sizeof(struct node));
    temp->data = data;
    temp->left = temp->right = NULL;
    return temp;
}
int height(struct node* root)
{
    if (root == NULL)
        return 0;
    int lh = height(root->left);
    int rh = height(root->right);
    return (lh > rh ? lh : rh) + 1;
}
void printLevel(struct node* root, int level)
{
    if (root == NULL)
        return;
    if (level == 1)
        printf("%d ", root->data);
    else
     {
        printLevel(root->left, level - 1);
        printLevel(root->right, level - 1);
    }
}
```

```
void bfs(struct node* root)
{
    int h = height(root);
    for (int i = 1; i <= h; i++)
        printLevel(root, i);
}

int main()
{
    struct node* root = newNode(10);
    root->left = newNode(20);
    root->right = newNode(30);
    root->left->left = newNode(40);
    root->left->right = newNode(50);
    root->right->left = newNode(60);
    printf("Breadth First Search (Level Order):\n");
    bfs(root);
    printf("\nMAC Address:24:6A:0E:DD:3A:0F\n");
    return 0;
}
```

```
Overwriting bfs.c
```

```
%%bash
gcc bfs.c -o bfs
./bfs
```

```
Breadth First Search (Level Order):
10 20 30 40 50 60
MAC Address:24:6A:0E:DD:3A:0F
```

```
%%writefile level_order.c
//24.Program to print level order in the tree
#include <stdio.h>
#include <stdlib.h>
struct node
{
    int data;
    struct node *left;
    struct node *right;
};
struct node* newNode(int data)
 {
    struct node* temp = (struct node*)malloc(sizeof(struct node));
    temp->data = data;
    temp->left = temp->right = NULL;
    return temp;
}
int height(struct node* root)
{
    if (root == NULL)
        return 0;
    int lh = height(root->left);
    int rh = height(root->right);
    return (lh > rh ? lh : rh) + 1;
}
void printLevel(struct node* root, int level)
{
    if (root == NULL)
        return;
    if (level == 1)
        printf("%d ", root->data);
    else {
        printLevel(root->left, level - 1);
        printLevel(root->right, level - 1);
    }
}
void levelOrder(struct node* root)
{
    int h = height(root);
    for (int i = 1; i <= h; i++)
        printLevel(root, i);
}
int main()
 {
    struct node* root = newNode(1);
```

```
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(4);
    root->left->right = newNode(5);
    printf("Level Order Traversal:\n");
    levelOrder(root);
    printf("\nMAC Address:24:6A:0E:DD:3A:0F\n");
    return 0;
}
```

```
Overwriting level_order.c
```

```
%%bash
gcc level_order.c -o level_order
./level_order
```

```
Level Order Traversal:
1 2 3 4 5
MAC Address:24:6A:0E:DD:3A:0F
```

```
%%writefile dfs_bfs_adj.c
//25.Program to construct DFS and BFS using adjacency list(stacks and queues)
#include <stdio.h>
#include <stdlib.h>
#define MAX 10
struct node
{
    int vertex;
    struct node *next;
};
struct graph
{
    int vertices;
    struct node *adjList[MAX];
};
int stack[MAX], top = -1;
int queue[MAX], front = 0, rear = -1;
struct node* createNode(int v)
 {
    struct node *newNode = (struct node*)malloc(sizeof(struct node));
    newNode->vertex = v;
    newNode->next = NULL;
    return newNode;
}
struct graph* createGraph(int v)
 {
    struct graph *g = (struct graph*)malloc(sizeof(struct graph));
    g->vertices = v;
    for (int i = 0; i < v; i++)
        g->adjList[i] = NULL;
    return g;
}
void addEdge(struct graph *g, int src, int dest)
{
    struct node *newNode = createNode(dest);
    newNode->next = g->adjList[src];
    g->adjList[src] = newNode;
    newNode = createNode(src);
    newNode->next = g->adjList[dest];
    g->adjList[dest] = newNode;
}
void push(int v)
{
  stack[++top] = v;
}
int pop()
{
  return stack[top--];
}
int isStackEmpty()
{
  return top == -1;
}
void enqueue(int v)
{
  queue[++rear] = v;
```

```
}
int dequeue()
{
  return queue[front++];
}
int isQueueEmpty()
{
  return front > rear;
 }
void DFS(struct graph *g, int start)
{
    int visited[MAX] = {0};
    printf("DFS Traversal: ");
    push(start);
    while (!isStackEmpty())
     {
        int v = pop();
        if (!visited[v])
        {
            printf("%d ", v);
            visited[v] = 1;
        }
        struct node *temp = g->adjList[v];
        while (temp)
        {
            if (!visited[temp->vertex])
                push(temp->vertex);
            temp = temp->next;
        }
    }
    printf("\n");
}
void BFS(struct graph *g, int start)
 {
    int visited[MAX] = {0};
    printf("BFS Traversal: ");
    enqueue(start);
    visited[start] = 1;
    while (!isQueueEmpty())
    {
        int v = dequeue();
        printf("%d ", v);
        struct node *temp = g->adjList[v];
        while (temp)
         {
            if (!visited[temp->vertex])
             {
                visited[temp->vertex] = 1;
                enqueue(temp->vertex);
            }
            temp = temp->next;
        }
    }
    printf("\n");
}
int main()
{
    int vertices = 5;
    struct graph *g = createGraph(vertices);
    addEdge(g, 0, 1);
    addEdge(g, 0, 2);
    addEdge(g, 1, 3);
    addEdge(g, 1, 4);
    DFS(g, 0);
    BFS(g, 0);
    printf("\nMAC Address:24:6A:0E:DD:3A:0F\n");
    return 0;
}
```

```
Overwriting dfs_bfs_adj.c
```

```
%%bash
gcc dfs_bfs_adj.c -o dfs_bfs_adj
./dfs_bfs_adj
```

```
DFS Traversal: 0 1 3 4 2
BFS Traversal: 0 2 1 4 3
```

```
MAC Address:24:6A:0E:DD:3A:0F
```

```
%%writefile mal_cal.c
//26.Program to check  wheather a junk/zero is initialized to calloc and malloc functions
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int n = 5;
    int *marr, *carr;
    marr = (int *)malloc(n * sizeof(int));
    carr = (int *)calloc(n, sizeof(int));
    printf("Values after malloc():\n");
    for (int i = 0; i < n; i++)
     {
        printf("%d ", marr[i]);
    }
    printf("\n\nValues after calloc():\n");
    for (int i = 0; i < n; i++)
    {
        printf("%d ", carr[i]);
    }
    free(marr);
    free(carr);
    printf("\nMAC Address:24:6A:0E:DD:3A:0F\n");
    return 0;
}
```

```
Writing mal_cal.c
```

```
%%bash
gcc mal_cal.c -o mal_cal
./mal_cal
```

```
Values after malloc():
0 0 0 0 0

Values after calloc():
0 0 0 0 0
MAC Address:24:6A:0E:DD:3A:0F
```

```
%%writefile circular_ll.c
//27.Program to perform operations of circular linked list
#include <stdio.h>
#include <stdlib.h>
struct node
 {
    int data;
    struct node *next;
};
struct node *last = NULL;
void insertBegin(int value)
 {
    struct node *newNode = (struct node *)malloc(sizeof(struct node));
    newNode->data = value;
    if (last == NULL)
    {
        last = newNode;
        last->next = last;
    }
    else
    {
        newNode->next = last->next;
        last->next = newNode;
    }
}
void insertEnd(int value)
 {
    struct node *newNode = (struct node *)malloc(sizeof(struct node));
    newNode->data = value;
    if (last == NULL)
    {
        last = newNode;
        last->next = last;
    }
```

```
    else
    {
        newNode->next = last->next;
        last->next = newNode;
        last = newNode;
    }
}
void deleteBegin()

 {
    if (last == NULL)
    {
        printf("List is empty\n");
        return;
    }
    struct node *temp = last->next;
    if (temp == last)
     {
        last = NULL;
    }
    else
     {
        last->next = temp->next;
    }
    printf("Deleted: %d\n", temp->data);
    free(temp);
}
void display()
{
    if (last == NULL)
     {
        printf("List is empty\n");
        return;
    }
    struct node *temp = last->next;
    printf("Circular Linked List: ");
    do
    {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    while (temp != last->next);
    printf("\n");
}

int main()
{
    insertBegin(10);
    insertBegin(20);
    insertEnd(30);
    insertEnd(40);
    display();
    deleteBegin();
    display();
    printf("\nMAC Address:24:6A:0E:DD:3A:0F\n");
    return 0;
}
```

```
Writing circular_ll.c
```

```
%%bash
gcc circular_ll.c -o circular_ll
./circular_ll
```

```
Circular Linked List: 20 10 30 40
Deleted: 20
Circular Linked List: 10 30 40

MAC Address:24:6A:0E:DD:3A:0F
```

```
%%writefile string_compare.c
//28.Program to compare 2 strings using built-in function
#include <stdio.h>
#include <string.h>
int main()
{
```

```
    char str1[50], str2[50];
    printf("Enter first string: ");
    scanf("%s", str1);
    printf("Enter second string: ");
    scanf("%s", str2);
    if (strcmp(str1, str2) == 0)
        printf("\nBoth strings are equal\n");
    else
        printf("Strings are not equal\n");
    printf("\nMAC Address:24:6A:0E:DD:3A:0F\n");
    return 0;
}
```

```
Overwriting string_compare.c
```

```
%%bash
gcc string_compare.c -o string_compare
printf "hello\nhello\n" | ./string_compare
```

```
Enter first string: Enter second string:
Both strings are equal

MAC Address:24:6A:0E:DD:3A:0F
```

```
%%writefile ll_middle.c
//29.Program to insertion and deletion of node in linked list
#include <stdio.h>
#include <stdlib.h>
struct node
{
    int data;
    struct node *next;
};
struct node *head = NULL;
void insertEnd(int value)
{
    struct node *newNode = (struct node *)malloc(sizeof(struct node));
    newNode->data = value;
    newNode->next = NULL;
    if (head == NULL)
    {
        head = newNode;
        return;
    }
    struct node *temp = head;
    while (temp->next != NULL)
        temp = temp->next;
    temp->next = newNode;
}
void insertMiddle(int value, int pos)
{
    if (pos <= 1 || head == NULL)
     {
        struct node *newNode = (struct node *)malloc(sizeof(struct node));
        newNode->data = value;
        newNode->next = head;
        head = newNode;
        return;
    }
    struct node *temp = head;
    for (int i = 1; i < pos - 1 && temp->next != NULL; i++)
        temp = temp->next;
    struct node *newNode = (struct node *)malloc(sizeof(struct node));
    newNode->data = value;
    newNode->next = temp->next;
    temp->next = newNode;
}
void deleteMiddle(int pos)
{
    if (head == NULL || pos <= 0)
    {
        printf("Invalid operation\n");
        return;
    }
    if (pos == 1)
    {
```

```
        struct node *del = head;
        head = head->next;
        printf("Deleted: %d\n", del->data);
        free(del);
        return;
    }
    struct node *temp = head;
    for (int i = 1; i < pos - 1 && temp->next != NULL; i++)
        temp = temp->next;
   if (temp->next == NULL)
   {
        printf("Position out of range\n");
        return;
    }
    struct node *del = temp->next;
    temp->next = del->next;
    printf("Deleted: %d\n", del->data);
    free(del);
}
void display()
 {
    struct node *temp = head;
    printf("Linked List: ");
    while (temp != NULL)
    {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}
int main()
{
    insertEnd(10);
    insertEnd(20);
    insertEnd(40);
    insertEnd(50);
    display();
    insertMiddle(30, 3);
    display();
    deleteMiddle(4);
    display();
    printf("\nMAC Address:24:6A:0E:DD:3A:0F\n");
    return 0;
}
```

```
Overwriting ll_middle.c
```

```
%%bash
gcc ll_middle.c -o ll_middle
./ll_middle
```

```
Linked List: 10 20 40 50
Linked List: 10 20 30 40 50
Deleted: 40
Linked List: 10 20 30 50

MAC Address:24:6A:0E:DD:3A:0F
```

```
%%writefile tree_traversal.c
//30.Program to constrct traversal binary tree
#include <stdio.h>
#include <stdlib.h>
struct node
 {
    int data;
    struct node *left;
    struct node *right;
};
struct node* newNode(int data)
{
    struct node *temp = (struct node*)malloc(sizeof(struct node));
    temp->data = data;
    temp->left = temp->right = NULL;
    return temp;
}
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()
{
    struct node *root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(4);
    root->left->right = newNode(5);
    printf("Inorder Traversal: ");
    inorder(root);
    printf("\nPreorder Traversal: ");
    preorder(root);
    printf("\nPostorder Traversal: ");
    postorder(root);
    printf("\nMAC Address:24:6A:0E:DD:3A:0F\n");
    return 0;
}
```

```
Overwriting tree_traversal.c
```

```
%%bash
gcc tree_traversal.c -o tree_traversal
./tree_traversal
```

```
Inorder Traversal: 4 2 5 1 3
Preorder Traversal: 1 2 4 5 3
Postorder Traversal: 4 5 2 3 1
MAC Address:24:6A:0E:DD:3A:0F
```

```
%%writefile circular_queue.c
//31.Program to check for modulo division ,queue full and queue empty for circular queues
#include <stdio.h>
#define SIZE 5
int cq[SIZE];
int front = -1, rear = -1;
int isFull()
{
    return (front == (rear + 1) % SIZE);
}
int isEmpty()
 {
    return (front == -1);
}
void enqueue(int value)
 {
    if (isFull())
    {
        printf("Queue is FULL\n");
        return;
    }
    if (isEmpty())
        front = rear = 0;
    else
        rear = (rear + 1) % SIZE;
    cq[rear] = value;
```

```
    printf("Inserted %d\n", value);
}
void dequeue()
 {
    if (isEmpty())
    {
        printf("Queue is EMPTY\n");
        return;
    }
    printf("Deleted %d\n", cq[front]);
    if (front == rear)
        front = rear = -1;
    else
        front = (front + 1) % SIZE;
}
void display()
{
    if (isEmpty())
     {
        printf("Queue is EMPTY\n");
        return;
    }
    printf("Circular Queue: ");
    int i = front;
    while (1)
    {
        printf("%d ", cq[i]);
        if (i == rear)
            break;
        i = (i + 1) % SIZE;
    }
    printf("\n");
}
int main()
{
    enqueue(10);
    enqueue(20);
    enqueue(30);
    enqueue(40);
    enqueue(50);
    display();
    dequeue();
    dequeue();
    display();
    enqueue(60);
    enqueue(70);
    display();
    printf("\nMAC Address:24:6A:0E:DD:3A:0F\n");
    return 0;
}
```

```
Writing circular_queue.c
```

```
%%bash
gcc circular_queue.c -o circular_queue
./circular_queue
```

```
Inserted 10
Inserted 20
Inserted 30
Inserted 40
Inserted 50
Circular Queue: 10 20 30 40 50
Deleted 10
Deleted 20
Circular Queue: 30 40 50
Inserted 60
Inserted 70
Circular Queue: 30 40 50 60 70

MAC Address:24:6A:0E:DD:3A:0F
```

```
%%writefile sparse_ll.c
//32.Program to construct sparse matrix using linked lists
#include <stdio.h>
#include <stdlib.h>
struct node
```

```
    {
        int row, col, val;
        struct node *next;
    };
    struct node *head = NULL;
    void insert(int r, int c, int v)
    {
        struct node *newNode = (struct node*)malloc(sizeof(struct node));
        newNode->row = r;
        newNode->col = c;
```